Kristu Jayanti College

AUTONOMOUS Bengaluru

Reaccredited A++ Grade by NAAC | Affiliated to Bengaluru North University

PROJECT REPORT ON

“ BuzzHub - Online Buzzer System ”

Project Report submitted in partial fulfilment of the requirements for the Software Engineering Mini Project for 2nd Semester.

MASTER OF COMPUTER APPLICATIONS (MCA)

Submitted by

Abhimanyu R ***23MCAA01***

Hemanth Kumar ***23MCAA26***

Under the guidance of

Dr. K. Prakash

Kristu Jayanti College
AUTONOMOUS Bengaluru
Reaccredited A++ Grade by NAAC | Affiliated to Bengaluru North University

CERTIFICATE OF COMPLETION

This is to certify that the project entitled " **BuzzHub - Online Buzzer System** " has been satisfactorily completed by **MADANAPALLI HEMANTH KUMAR** , **23MCAA26** in partial fulfilment of the requirements for the ***Software Engineering Mini Project*** with course code **MCC2P2B21**, for the 2nd Semester MCA course during the academic year 2023 - 2024 as prescribed by Bangalore University.

Internal Guide ***Head of the Department***

Valued by Examiners

1: ____________________ ***Centre:*** **Kristu Jayanti College**

2: ____________________ ***Date:***

Kristu Jayanti College

AUTONOMOUS Bengaluru

Reaccredited A++ Grade by NAAC | Affiliated to Bengaluru North University

DECLARATION

I, MADANAPALLI HEMANTH KUMAR 23MCAA26 hereby declare that the project work entitled "BuzzHub - Online Buzzer System" is an original project work carried out by me, under the guidance of Dr. K. Prakash

This project work has not been submitted earlier either to any University / Institution or any other body for the fulfillment of the requirement of a course of study.

Signature

MADANAPALLI HEMANTH KUMAR

Bengaluru

Date:

ACKNOWLEDGEMENT

The success of the project depends upon the efforts invested. It’s my duty to acknowledge and thank the individuals who has contributed to the successful completion of the project.

I take this opportunity to express my profound and wholehearted thanks to **Rev. Fr. Dr. Augustine George,** our principal, for providing us his constant guidance and support. I would also like to thank **Rev. Fr. Lijo P Thomas**, our Vice-Principal and Chief Finance Officer, for providing ample facilities made to undergo my project successfully.

I express my deep sense of gratitude and sincere thanks to our Head of the Department **Dr. R. Kumar** for his enormous guidance, motivation, also giving us the essential support.

I feel immense pleasure to thank my respected guide **Dr. K. Prakash** for sustaining Interest and providing dynamic guidance in aiding me to complete this project immaculately and impeccably and for being the source of my strength and confidence.

It is my duty to express my thanks to all Teaching and Non-Teaching Staff members of Computer science department who offered me help directly or indirectly by their suggestions.
The successful completion of my project would not have been possible without my parent’s sacrifice, and guidance. I take this opportunity to thank everyone for their continuous encouragement.

Last but not the least I convey my thankfulness to all my friends who were with me to share my happiness, agony and in pointing out the error and help in improve the project.

TABLE OF CONTENTS

1. INTRODUCTION

"BuzzHub - Online Buzzer System" is a dynamic web application crafted to elevate the experience of quiz games, trivia nights, and educational competitions. Its goal is to unite participants in a virtual realm, enabling real-time competition and the demonstration of quick thinking and knowledge through a buzzer-based gameplay. Built with React.js, Next.js, Socket.io, and a blend of CSS, HTML, and JavaScript, BuzzHub promises an immersive and engaging platform for interactive events.

1.1 PROBLEM DEFINITION

The conventional approach to buzzer systems has been confined to physical setups, limiting participation to localized audiences and impeding scalability and real-time interaction. This inherent restriction poses significant challenges for organizers aspiring to host engaging competitions with a global reach. Moreover, the absence of online accessibility deprives enthusiasts worldwide of the opportunity to participate and engage in spirited competition, underscoring the pressing need for an innovative solution.

Recognizing these challenges, BuzzHub emerges as a pioneering platform addressing the need for an engaging and interactive environment to host quiz games, trivia nights, and educational competitions in a virtual space. Traditional methods of conducting such events in person often encounter hurdles such as geographical limitations, scheduling conflicts, or health concerns, making them less feasible or accessible. BuzzHub aims to transcend these barriers by providing a seamless online solution where participants can engage in real-time competition, showcasing their knowledge and quick thinking through buzzer-based gameplay.

1.2 SCOPE OF THE PROJECT

- The scope of BuzzHub, an online buzzer system, encompasses the development and deployment of software designed to host engaging quiz games, trivia nights, and educational competitions in a virtual environment. The system aims to facilitate real-time interaction among participants, allowing them to showcase their knowledge and quick thinking through buzzer-based gameplay.

- BuzzHub provides an intuitive and user-friendly online platform accessible to users worldwide. Whether joining existing games or creating new ones, participants navigate seamlessly through the platform, fostering a vibrant community of trivia enthusiasts.

- Central to BuzzHub's mission is enabling real-time interaction among participants, regardless of their geographical locations. Users engage in dynamic gameplay, fueling excitement and camaraderie throughout each competition.

- The platform is built on a robust infrastructure capable of handling concurrent users and diverse game scenarios. BuzzHub ensures scalability and accessibility for both organizers and participants, accommodating the growing demand for buzzer-based competitions while maintaining optimal performance.
- BuzzHub implements industry best practices for secure authentication, safeguarding user data and ensuring a safe gaming environment. The platform prioritizes user privacy and security, instilling confidence and trust among its user base.

- Overall, BuzzHub is efficient, accurate, flexible, and user-friendly, enhancing the gaming experience for participants and organizers alike.

MODULES IN THE PROJECT

• Game Management Module
• UI/UX Components Module
• Sound Management Module
• Data Persistence Module
• Host Module
• User Module

GAME MANAGEMENT MODULE: This module is responsible for managing game sessions, tracking game progress, and enforcing game rules. It utilizes the "boardgame.io" library for these functionalities.

UI/UX COMPONENTS MODULE: This module includes UI libraries and components for building an intuitive and engaging user interface. It incorporates React, React Bootstrap.

SOUND MANAGEMENT MODULE: This module manages the playback of buzzer sounds and other audio cues during gameplay. It utilizes the "howler" library to handle audio playback and management of sound effects.

DATA PERSISTENCE MODULE: This module handles the storage and retrieval of data related to game sessions, user profiles, and other system entities. It uses the "axios" library for making HTTP requests, commonly used for handling data persistence tasks such as sending and receiving data from a server.

HOST MODULE: The host module facilitates the hosting of game sessions, allowing users to create and manage game rooms. It includes functionalities for starting games, controlling game settings, and managing player access.

USER MODULE: This module enables players to join game sessions using room IDs provided by hosts. It handles player authentication, room joining, and participation in buzzer-based gameplay within the BuzzHub platform.

2. SYSTEM STUDY

2.1 STUDY OF EXISTING SYSTEM

The problem with traditional buzzer systems lies in their reliance on physical setups, which limits audience reach and scalability. These systems lack real-time interaction features and are confined to local participation due to limited online accessibility. Additionally, manual management is required, leading to inefficiencies and potential errors. Key shortcomings of existing buzzer systems include:

- Limited audience reach and scalability due to physical setups.
- Lack of real-time interaction features and online accessibility.
- Requirement for manual management, leading to inefficiencies and potential errors.
- Absence of real-time data analysis and participant engagement features

In summary, the existing buzzer systems underscore the need for a modernized approach to overcome these limitations and provide an enhanced user experience.

2.2 FEASIBILITY STUDY

A feasibility study is an analysis of how successfully a project can be completed, accounting for factors that affect it such as economic, technological, legal, and scheduling factors. Project managers use feasibility studies to determine potential positive and negative outcomes of a project before investing a considerable amount of time and money into it.

A feasibility study tests the viability of an idea, a project or even a new business. The goal of a feasibility study is to place emphasis on potential problems that could occur if a project is pursued and determine if, after all significant factors are considered, the project should be pursued.

For the **"BuzzHub - Online Buzzer System"** project, a feasibility study would be conducted to evaluate its viability. Here are the components of feasible study:

- Technical Feasibility

- Operational Feasibility
- Schedule Feasibility

Feasibility studies are almost always conducted where large sums are at stake. Also called as feasibility analysis. Every project is feasible for given unlimited resources and infinitive time. Feasibility study is an evaluation of the proposed system regarding its workability, impact on the organization, ability to meet the user needs and effective use of resources. Thus, when a new application is proposed it normally goes through a feasibility study before it is approved for development. Feasibility and risk analysis are related in many ways. The feasibility analysis in this project has been discussed below based on the above-mentioned components of feasibility.

Technical feasibility

- Assessing the technical capabilities required for developing the online buzzer system, including software and hardware requirements.
- Evaluating the compatibility of chosen technologies and platforms for implementing the system, such as web development frameworks and hosting services.
- Determining the feasibility of implementing desired features and functionalities within the technical constraints

Operational feasibility

- Analyzing how the online buzzer system will integrate into existing operational processes and workflows.
- Assessing the ease of use and user experience of the system for both administrators and participants.
- Evaluating the system's scalability and ability to handle a growing user base and increasing demand for buzzer-based competitions.

Schedule feasibility

- Estimating the time and resources required for developing, testing, and deploying the online buzzer system.

- Identifying potential risks and challenges that could impact the project timeline, such as technical setbacks or resource constraints.
- Developing a realistic project schedule with achievable milestones and deadlines.

2.3 PROPOSED SYSTEM

The proposed BuzzHub - Online Buzzer System offers several advantages over existing buzzer systems:

• Increased Accessibility: By transitioning to an online platform, BuzzHub eliminates geographical limitations, allowing participants from anywhere with internet access to join buzzer-based competitions.

• Enhanced Real-Time Interaction: With features such as real-time game updates and instant feedback, BuzzHub fosters dynamic engagement among participants, creating a more immersive and interactive gaming experience.

• Scalability: Unlike traditional buzzer systems limited by physical setups, BuzzHub is designed to accommodate a large number of concurrent users and scale seamlessly to meet growing demand.

• Improved Efficiency: By automating processes such as game management, scoring, and player communication, BuzzHub streamlines the gaming experience, reducing manual effort and minimizing potential errors.

• Enhanced User Experience: With intuitive user interfaces, and responsive design, BuzzHub prioritizes user experience, making it easy for both organizers and participants to navigate and enjoy the platform.

Overall, the proposed BuzzHub system revolutionizes buzzer-based competitions by offering a modern, accessible, and efficient platform that caters to the needs of organizers and participants alike.

3. SYSTEM DESIGN

In the design phase the architecture is established. This phase starts with the requirement document delivered by the requirement phase and maps the requirements into an architecture. The architecture defines the components, their interfaces, and behaviors. The deliverable design document is architecture. The design document describes a plan to implement the requirements. This phase represents the “how” phase

Details on computer programming languages and environments, machines, packages, application architecture, distributed architecture layering, memory size, platform, algorithms, data structures, global type definitions, interfaces, and many other engineering details are established.

The design may include the usage of existing components. Analyzing the trade-offs of necessary complexity allows for many things to remain simple which, in turn, will eventually lead to a higher quality product. The architecture team also converts the typical scenarios into a test plan. In our approach, the team, given a complete requirement document, must also indicate critical priorities for the implementation team.

A critical implementation priority leads to a task that has to be done right. If it fails, the product fails. If it succeeds, the product might succeed. At the very least, the confidence level of the team producing a successful product will increase. This will keep the implementation team focused. Exactly how this information is conveyed is a skill based on experience more than a science based on fundamental foundations. System design is the process of defining the architecture components, modules, interfaces, and data for a system to satisfy specified requirements.

Systems design could be seen as the application of systems theory to product development. There is some overlap with the disciplines of systems analysis, systems architecture, and systems engineering. If the broader topic of product development blends the perspective of marketing, design, and manufacturing into a single approach to product development," then design is the act of taking the marketing information and creating the design of the product to

be manufactured. Systems design is therefore the process of defining and developing systems to satisfy specified requirements of the user.
Until the 1990s, systems design had a crucial and respected role in the data processing industry. In the 1990s, standardization of hardware and software resulted in the ability to build modular systems. The increasing importance of software running on generic platforms has enhanced the discipline of software engineering.

Object-oriented analysis and design methods are becoming the most widely used methods for computer systems design. The UML has become the standard language in object-oriented analysis and design. It is widely used for modelling software systems and is increasingly used for high designing non software systems and organizations.

ARCHITECTURAL DESIGN

The architectural design of a system emphasizes the design of the system architecture that describes the structure, behavior, and more views that system and analysis.

LOGICAL DESIGN

The logical design of a system pertains to an abstract representation of the data flows, inputs, and outputs of the system. This is often conducted via modelling, using an over-abstract (and sometimes graphical) model of the actual system. In the context of systems, designs are included. Logical design includes entity-relationship diagrams (ER diagrams).

PHYSICAL DESIGN

The physical design relates to the actual input and output processes of the system. This is explained in terms of how data is input into a system, how it is verified/authenticated, how it is processed, and how it is displayed. In physical design, the following requirements about the system are decided.

Input requirement,
Output requirements,
Storage requirements, Processing requirements,
System control and backup or recovery.

Put another way, the physical portion of system design can generally be broken down into three sub- tasks:

- User Interface Design.
- Data Design.
- Process Design

User Interface Design is concerned with how users add information to the system and with how the system presents information back to them. Data Design is concerned with how the data is represented and stored within the system.

Finally, Process Design is concerned with how data moves through the system, and with how and where it is validated, secured and/or transformed as it flows into, through and out of the system.

At the end of the system design phase, documentation describing the three sub-tasks is produced and made available for use in the next phase. Physical design, in this context, does not refer to the tangible physical design of an information system.

To use an analogy, a personal computer's physical design involves input via a keyboard, processing within the CPU, and output via a monitor, printer, etc.

It would not concern the actual layout of the tangible hardware, which for a PC would be a monitor, CPU, motherboard, hard drive, modems, video/graphics cards, USB slots, etc.

It involves a detailed design of a user and a product database structure processor and a control processor. The H/S personal specification is developed for the proposed system.

3.1 ER DIAGRAM

An Entity Relationship (ER) Diagram is a type of flowchart that illustrates how “entities” such as people, objects or concepts relate to each other within a system. ER Diagrams are most often used to design or debug relational databases in the fields of software engineering, business information systems, education, and research. Also known as ERDs or ER Models, they use a defined set of symbols such as rectangles, diamonds, ovals and connecting lines to depict the interconnectedness of entities, relationships, and their attributes. They mirror grammatical structure, with entities as nouns and relationships as verbs.

ER diagrams are related to data structure diagrams (DSDs), which focus on the relationships of elements within entities instead of relationships between entities themselves. ER diagrams also are often used in conjunction with data flow diagrams (DFDs), which map out the flow of information for processes or systems.

Uses of entity relationship diagrams:

DATABASE DESIGN: ER diagrams are used to model and design relational databases, in terms of logic and business rules (in a logical data model) and in terms of the specific technology to be implemented (in a physical data model.) In software engineering, an ER diagram is often an initial step in determining requirements for an information systems project. It’s also later used to model a particular
database or databases. A relational database has an equivalent relational table and can potentially be expressed that way as needed.

DATABASE TROUBLESHOOTING: ER diagrams are used to analyze existing databases to find and resolve problems in logic or deployment. Drawing the diagram should reveal where it’s going wrong.

BUSINESS INFORMATION SYSTEMS: The diagrams are used to design or analyze relational databases used in business processes. Any business process that uses fielded data involving entities, actions and interplay can potentially benefit from a relational database. It can streamline processes, uncover information more easily and improve results.

BUSINESS PROCESS RE-ENGINEERING (BPR): ER diagrams help in analyzing databases used in business process re-engineering and in modeling a new database setup.

EDUCATION: Databases are today’s method of storing relational information for educational purposes and later retrieval, so ER Diagrams can be valuable in planning those data structures.

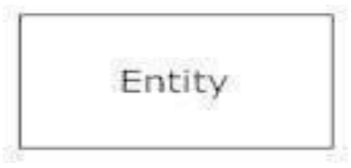
RESEARCH: Since so much research focuses on structured data, ER diagrams can play a key role in setting up useful databases to analyze the data.

THE COMPONENTS AND FEATURES OF AN ER DIAGRAM

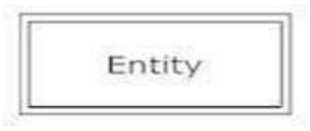
ER Diagrams are composed of entities, relationships, and attributes. They also depict cardinality, which defines relationships in terms of numbers.

ENTITY

A definable thing such as a person, object, concept, or event that can have data stored about it. Think of entities as nouns. Examples: a member , student, car or product. Typically shown as a rectangle.

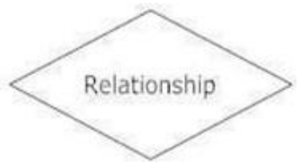


A weak entity is an entity that must defined by a foreign key relationship with another entity as itcannot be uniquely identified by its own attributes alone.



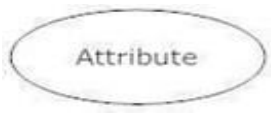
ACTIONS

Actions are represented by diamond shapes, show how two entities share information in the database. In some cases, entities can be self-linked. For example, employees can supervise other employees.

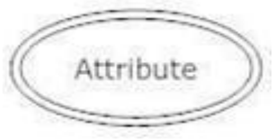


ATTRIBUTES

Attributes which are represented by ovals. A key attribute is the unique, distinguishing characteristic of the entity. For example, an employee's social security number might be the employee's key attribute. A multivalued attribute can have more than one value. For example, an employee entity can have multiple skill values. A derived attribute is based on another attribute. For example, an employee's monthly salary is based on the employee's annual salary.



Multivalued attribute can have more than one value. For example, an employee entity can have multiple skill values.



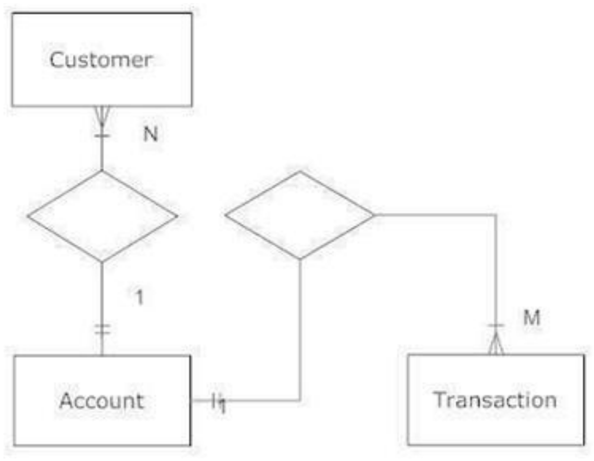
Derived attribute is based on another attribute. For example, an employee's monthly salary is based on the employee's annual salary

CONNECTING LINES

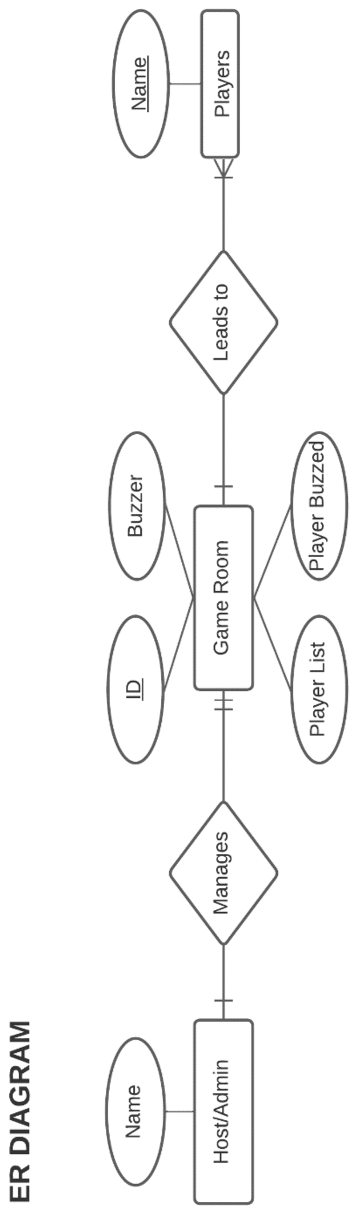
The solid lines that connect attributes to show the relationships of entities in the diagram.

CARDINALITY

It specifies how many instances of an entity relate to one instance of another entity. Ordinarily is also closely linked to cardinality. While cardinality specifies the occurrences of a relationship, ordinarily describes the relationship as either mandatory or optional. In other words, cardinality specifies the maximum number of relationships and ordinarily specifies the absolute minimum number of relationships.



ER DIAGRAM


ER DIAGRAM
Name
Host/Admin
Manages
ID
Buzzer
Game Room
Player List
Player Buzzed
Leads to
Name
Players

3.2 DATA FLOW DIAGRAM (DFD)

Data flow diagrams are also known as bubble charts. DFD is a designing tool used in the top-down approach to Systems Design. This context-level DFD is next "exploded", to produce a Level 1 DFD that shows some of the detail of the system being modeled. The Level 1 DFD shows how the system is divided into sub-systems (processes), each of which deals with one or more of the data flows to or from an external agent, and which together provide all of the functionality of the system as a whole. It also identifies internal data stores that must be present in order for the system to do its job and shows the flow of data between the various parts of the system.

Data flow diagrams are one of the three essential perspectives of the structured-systems analysis and design method. The sponsor of a project and the end users will need to be briefed and consulted throughout all stages of a system's evolution. With a data flow diagram, users can visualize how the system will operate, what the system will accomplish, and how the system will be implemented. The old system's dataflow diagrams can be drawn up and compared with the new system's data flow diagrams to draw comparisons to implement a more efficient system. Data flow diagrams can be used to provide the end user with a physical idea of where the data they input ultimately influences the structure of the whole. In the course of developing a set of levelled data flow diagrams the analyst/designer is forced to address how the system may be decomposed into component sub-systems.

There are different notations to draw data flow diagrams, defining different visual representations for processes, data stores, data flow, and external entities.

PHYSICAL VS. LOGICAL DFD

A logical DFD captures the data flows that are necessary for a system to operate. It describes the processes that are undertaken, the data required and produced by each process, and the stores needed to hold the data. On the other hand, a physical DFD shows how the system is implemented, either at the moment (Current Physical DFD), or how the designer intends it to be in the future (Required Physical DFD). Thus, a Physical DFD may be used to describe the set of data items that appear on each piece of paper that move around an office, and the fact that a particular set of pieces of paper are stored together in a filing cabinet. It is quite possible that a Physical DFD will include references to data that are duplicated, or redundant, and that the data stores, if implemented as a set, would constitute an unnormalized (or deformalized) relational database. In contrast, a Logical DFD attempts to capture the data flow aspects of a system in a form that has neither redundancy nor duplication.

HISTORY

The original developer of structured design, based on Martin and Estrin's "Data Flow Graph" model of computation. Starting in the 1970s, data flow diagrams (DFD) became a popular way to visualize the major steps and data involved in software system processes. DFDs were usually used to show data flow in a computer system, although they could in theory be applied. DFD were useful to document the major data flows or to explore a new high-level design in terms of data flow.

DATA FLOW

Represented by a unidirectional arrow. Data Flows show how data is moved through the System. Data Flows are labelled with a description of the data that is being passed through it. In our course, we need to understand and be able to draw 2 types of Data Flow Diagrams, they are Level-0 and Level 1 DFD ‘s. In this blog, I will hopefully make it easier to understand the differences between the two types of DFD ‘s and help understand how to draw a DFD for the exam. Firstly, I will look at level-0 DFD ‘s and give an example. Then I will look at Level 1 DFD ‘s and give an example. A level-0 DFD is the most basic form of DFD. It aims to show how the entire system works at a glance. There is only one process in the system and all the data flows either into or out of this process. Level-0 DFD ‘s demonstrates the interactions between the process and external entities. They do not contain Data Stores. When drawing Level-0 DFD ‘s, we must first identify the process, all the external entities and all the data flows. We must also state any assumptions we make about the system. It is advised that we draw the process in the middle of the page. We then draw our external entities in the corners and finally connect our entities to our process with the data flows.

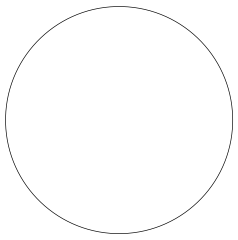
DATA FLOW SYMBOLS AND THEIR MEANINGS

DFDs only involve four symbols.

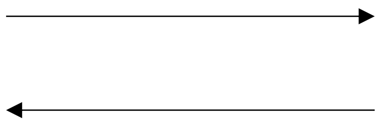
They are:

- Process
- Data flow
- Data Store
- External entity

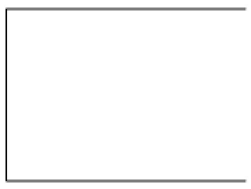
PROCESS: Transform of incoming data flow(s) to outgoing flow(s).



DATA FLOW: Movement of data in the system.



DATA STORE: Data repositories for data that are not moving. It may be as simple as a buffer or a queue or a sophisticated as a relational database.



EXTERNAL ENTITY: Sources of destinations outside the specified system boundary.

STEPS TO CONSTRUCT DATA FLOW DIAGRAMS

Four steps are commonly used to construct a DFD

- Process should be named and numbered for easy reference. Each name should be representative of the process.
- The destination of flow is from top to bottom and from left to right.
- When a process is exploded in to lower level details they are numbered.
- The names of data stores, sources and destinations are written in capital letters.

RULES FOR CONSTRUCTING A DATA FLOW DIAGRAM

- Arrows should not cross each other.
- Squares, circles, and files must bear names.
- Decomposed data flow squares and circles can have same names.
- Draw all data flow around the outside of the diagram

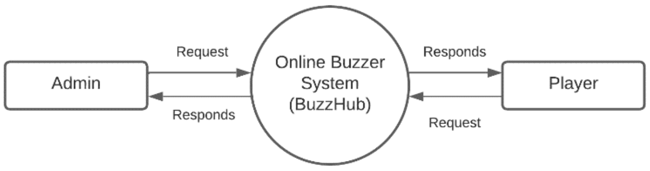
DFD levels

A data flow diagram can dive into progressively more detail by using levels and layers, zeroing in on a particular piece. DFD levels are numbered 0, 1 or 2, and occasionally go to even Level 3 or beyond. The necessary level of detail depends on the scope of what you are trying to accomplish.

DFD Level 0 is also called a Context Diagram. It's a basic overview of the whole system or process being analyzed or modelled. It's designed to be an at-a-glance view, showing the system as a single high-level process, with its relationship to external entities. It should be easily understood by a wide audience, including stakeholders, business analysts, data analysts and developers.

DFD Level 1 provides a more detailed breakout of pieces of the Context Level Diagram. You will highlight the main functions carried out by the system, as you break down the high-level process of the Context Diagram into its sub processes.

LEVEL 0 DFD


Admin
Request
Responds
Online Buzzer System (BuzzHub)
Responds
Request
Player

3.3 GANTT CHART

A Gantt chart is a type of bar chart, devised by Henry Gantt in the 1910s, that illustrates a project schedule. Gantt charts illustrate the start and finish dates of the terminal elements and summary elements of a project. Terminal elements and summary elements comprise the work breakdown structure of the project. Modern Gantt charts also show the dependency (i.e., precedence network) relationships between activities.

Historical Development

The first known tool of this type was developed in 1896 by Karol Adamiecki, who called it a Harmon gram. Adamiecki did not publish his chart until 1931, however, and only in Polish, which limited both its adoption and recognition of his authorship. The chart is named after Henry Gantt (1861–1919), who designed his chart around the years 1910–1915. One of the first major applications of Gantt charts was by the United States during World War I, at the instigation of General William Crosier in the 1980s, personal computers allowed widespread creation of complex and elaborate Gantt charts. The first desktop applications were intended mainly for project managers and project schedulers. With the advent of the Internet and increased collaboration over networks at the end of the 1990s, Gantt charts became a common feature of web-based applications, including collaborative groupware.

GANTT CHART BENEFITS

CLARITY:

One of the biggest benefits of a Gantt chart is the tool's ability to boil down multiple tasks and timelines into a single document. Stakeholders throughout an organization can easily understand where teams are in a process while grasping the ways in which independent elements come together toward project completion.

COMMUNICATION:

Teams can use Gantt charts to replace meetings and enhance other status updates. Simply clarifying chart positions offers an easy, visual method to help team members understand task progress.

MOTIVATION:

Some teams or team members become more effective when faced with a form of external motivation. Gantt charts offer teams the ability to focus work at the front of a task timeline, or at the tail end of a chart segment. Both types of team members can find Gantt charts meaningful as they plug their own work habits into the overall project schedule.

COORDINATION:

For project managers and resource schedulers, the benefits of a Gantt chart include the ability to sequence events and reduce the potential for overburdening team members. Some project managers even use combinations of charts to break down projects into more manageable sets of tasks.

CREATIVITY:

Sometimes, a lack of time or resources forces project managers and teams to find creative solutions. Seeing how individual tasks intertwine on Gantt charts often encourages new partnerships and collaborations that might not have evolved under traditional task assignment systems.

TIME MANAGEMENT:

Most managers regard scheduling as one of the major benefits of Gantt charts in a creative environment. Helping teams understand the overall impact of project delays can foster stronger collaboration while encouraging better task organization.

FLEXIBILITY:

Whether you use Excel to generate Gantt charts or you load tasks into a more precise chart generator, the ability to issue new charts as your project evolves lets you react to unexpected changes in project scope or timeline. While revising your project schedule too frequently can eliminate some of the other benefits of Gantt charts, offering a realistic view of a project can help team members recover from setbacks or adjust to other changes.

MANAGEABILITY:

For project managers handling complex assignments, like software publishing or event planning, the benefits of Gantt charts include externalizing assignments. By visualizing all of the pieces of a project puzzle, managers can make more focused, effective decisions about resources and timetables.

EFFICIENCY:

Another one of the benefits of Gantt charts is the ability for teams members to leverage each other’s deadlines for maximum efficiency. For instance, while one team member waits on the outcome of three other tasks before starting a crucial piece of the assignment, he or she can perform other project tasks. Visualizing resource usage during projects allows managers to make better use of people, places, and things.

ACCOUNTABILITY:

When project teams face major organizational change, documenting effort and outcomes becomes crucial to career success. Using Gantt charts during critical projects allows both project managers and participants to track team progress, highlighting both big wins and major failures during professional review periods; team members who frequently exceed expectations can leverage this documentation into larger raises or bonuses.

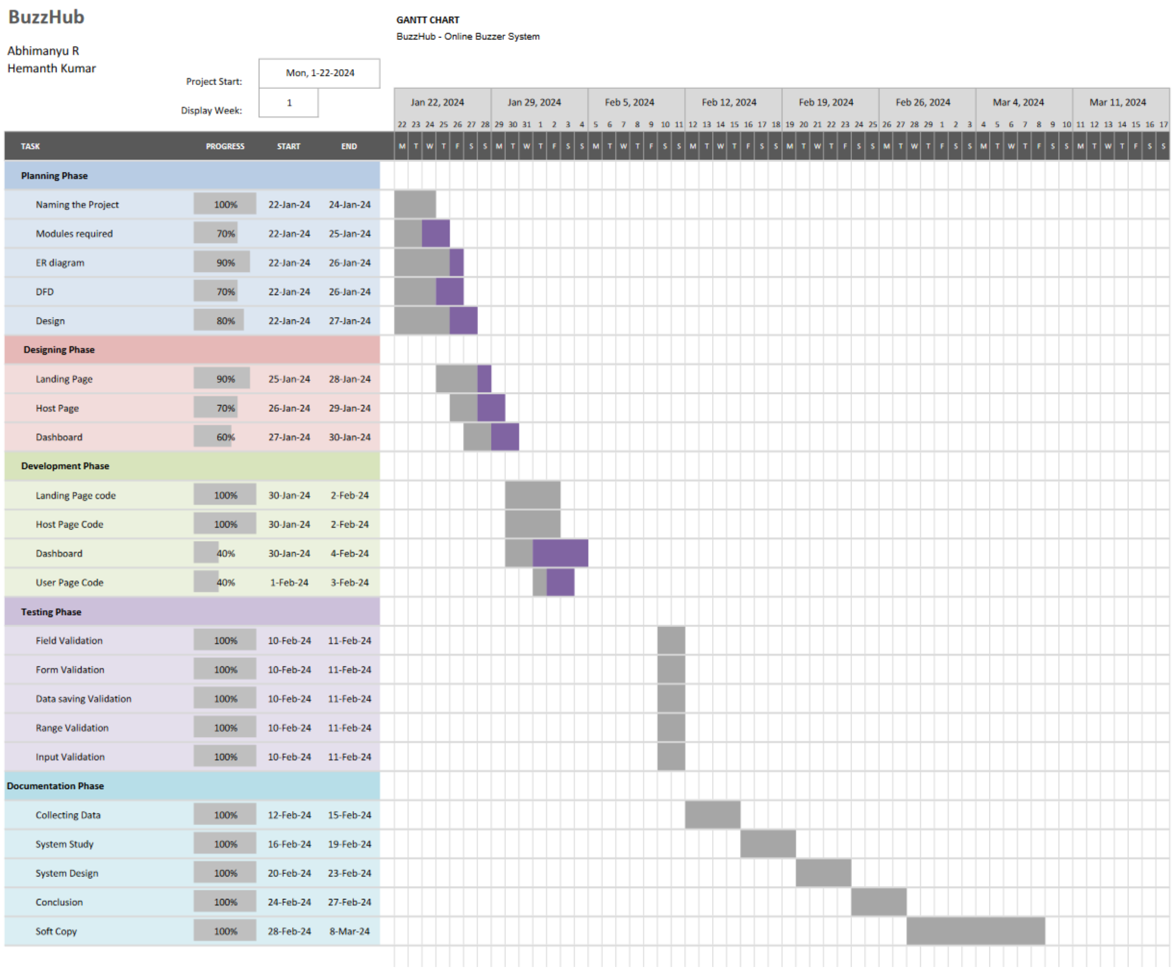
GANTT CHART IMPORTANCE

The project's summary and terminal elements, which combine to form the project's internal structure, are shown on the Gantt chart. Many charts will also depict the precedence rankings and dependencies of various tasks within the project. The charts can illustrate the start and finish project terminal elements in project management. It can also show summary elements and terminal dependencies. The smallest task tracked as part of the project effort is known as a terminal element. Gantt chart represents the tasks in most modern project scheduling packages. However other management applications use simpler communication tools such as message boards, to-do lists and simple scheduling etc., therefore, they do not use Gantt charts as heavily.

The way to create this chart begins by determining and listing the necessary activities. Next, sketch out how you expect the chart to look. List which items depend on others and what activities take place when.

For each activity, list how many man-hours it will require, and who is responsible. Lastly, determine the throughput time.

This technique's primary advantage is its good graphical overview that is easy to understand for nearly all project participants and stakeholders. Its primary disadvantage is its limited applicability for many projects since projects are often more complex than can be effectively communicated with this chart.



BuzzHub

Abhimanyu R
Hemanth Kumar

Project Start: Mon, 1-22-2024
Display Week: 1

GANTT CHART
BuzzHub - Online Buzzer System

TASK	PROGRESS	START	END
Planning Phase			
Naming the Project	100%	22-Jan-24	24-Jan-24
Modules required	70%	22-Jan-24	25-Jan-24
ER diagram	90%	22-Jan-24	26-Jan-24
DFD	70%	22-Jan-24	26-Jan-24
Design	80%	22-Jan-24	27-Jan-24
Designing Phase			
Landing Page	90%	25-Jan-24	28-Jan-24
Host Page	70%	26-Jan-24	29-Jan-24
Dashboard	60%	27-Jan-24	30-Jan-24
Development Phase			
Landing Page code	100%	30-Jan-24	2-Feb-24
Host Page Code	100%	30-Jan-24	2-Feb-24
Dashboard	40%	30-Jan-24	4-Feb-24
User Page Code	40%	1-Feb-24	3-Feb-24
Testing Phase			
Field Validation	100%	10-Feb-24	11-Feb-24
Form Validation	100%	10-Feb-24	11-Feb-24
Data saving Validation	100%	10-Feb-24	11-Feb-24
Range Validation	100%	10-Feb-24	11-Feb-24
Input Validation	100%	10-Feb-24	11-Feb-24
Documentation Phase			
Collecting Data	100%	12-Feb-24	15-Feb-24
System Study	100%	16-Feb-24	19-Feb-24
System Design	100%	20-Feb-24	23-Feb-24
Conclusion	100%	24-Feb-24	27-Feb-24
Soft Copy	100%	28-Feb-24	8-Mar-24

3.4 INPUT/OUTPUT DESIGN

Header:

```
import React from 'react';
import { Navbar } from 'react-bootstrap';
import { isNil } from 'lodash';
import { useHistory } from 'react-router';
import { leaveRoom } from '../lib/endpoints';

function Logo({ size = 25 }) {
  return (
    <svg
      width={size}
      height={size}
      viewBox="0 0 95 95"
      fill="none"
      xmlns="http://www.w3.org/2000/svg"
    >
      <circle cx="20" cy="20" r="20" fill="#5bd826" />
      <circle cx="75" cy="20" r="20" fill="#348DF5" />
      <circle cx="20" cy="75" r="20" fill="#348DF5" />
      <circle cx="75" cy="75" r="20" fill="#d0021b" />
    </svg>
  );
}

export default function Header({
  auth = {},
  clearAuth,
  sound = null,
  setSound,
}) {
  const history = useHistory();

  // leave current game
  async function leave() {
    try {
      await leaveRoom(auth.roomID, auth.playerID, auth.credentials);
      clearAuth();
      history.push('/');
    } catch (error) {
      console.log('leave error', error);
      clearAuth();
      history.push('/');
    }
  }

  return (
    <header>
      <Navbar>
        <Navbar.Brand>
          <Logo /> BuzzHub :)
        </Navbar.Brand>
              <div className="nav-buttons">

          {!isNil(sound) ? (
            <button className="text-button" onClick={() => setSound()}>
              {sound ? 'Turn off sound' : 'Turn on sound'}
            </button>
          ) : null}
          {clearAuth ? (
            <button className="text-button" onClick={() => leave()}>
              Leave game
            </button>
          ) : null}
        </div>
      </Navbar>
    </header>
  );
}

import React from 'react';
import { Container } from 'react-bootstrap';

export function FooterSimple() {
  return (
    <div id="footer-simple">
      Full Code{' '}
      <a
        target="_blank"
        rel="noopener noreferrer"
        href="https://github.com/abhimanyurajeesh/BuzzHub"
      >
        Click here
      </a>
    </div>
  );
}

/**
 * Footer component
 * @param {bool} mobileOnly - only display on mobile devices, <768 px
 */
export default function Footer({ mobileOnly = false }) {
  return (
    <footer className={mobileOnly ? 'd-block d-md-none' : null}>
      <Container>
        <div>
          Full Code{' '}
          <a
            target="_blank"
            rel="noopener noreferrer"
            href="https://github.com/abhimanyurajeesh/BuzzHub"
          >
            Click here
          </a>
        </div>
      </Container>
    </footer>
  );
}

import React, { useState, useEffect, useRef } from 'react';
import { get, some, values, sortBy, orderBy, isEmpty, round } from 'lodash';
import { Howl } from 'howler';
import { AiOutlineDisconnect } from 'react-icons/ai';
import { Container } from 'react-bootstrap';
import Header from '../components/Header';

export default function Table(game) {
```

```
const [loaded, setLoaded] = useState(false);
const [buzzed, setBuzzer] = useState(
  some(game.G.queue, (o) => o.id === game.playerID)
);
const [lastBuzz, setLastBuzz] = useState(null);
const [sound, setSound] = useState(false);
const [soundPlayed, setSoundPlayed] = useState(false);
const buzzButton = useRef(null);
const queueRef = useRef(null);

const buzzSound = new Howl({
  src: [
    `${process.env.PUBLIC_URL}/shortBuzz.webm`,
    `${process.env.PUBLIC_URL}/shortBuzz.mp3`,
  ],
  volume: 0.5,
  rate: 1.5,
});

const playSound = () => {
  if (sound && !soundPlayed) {
    buzzSound.play();
    setSoundPlayed(true);
  }
};

useEffect(() => {
  console.log(game.G.queue, Date.now());
  // reset buzzer based on game
  if (!game.G.queue[game.playerID]) {
    // delay the reset, in case game state hasn't reflected your buzz yet
    if (lastBuzz && Date.now() - lastBuzz < 500) {
      setTimeout(() => {
        const queue = queueRef.current;
        if (queue && !queue[game.playerID]) {
          setBuzzer(false);
        }
      }, 500);
    } else {
      // immediate reset, if it's been awhile
      setBuzzer(false);
    }
  }

  // reset ability to play sound if there is no pending buzzer
  if (isEmpty(game.G.queue)) {
    setSoundPlayed(false);
  } else if (loaded) {
    playSound();
  }

  if (!loaded) {
    setLoaded(true);
  }

  queueRef.current = game.G.queue;
}, [game.G.queue]);

const attemptBuzz = () => {
  if (!buzzed) {
    playSound();
```

```
      game.moves.buzz(game.playerID);
      setBuzzer(true);
      setLastBuzz(Date.now());
    }
  };

  // spacebar will buzz
  useEffect(() => {
    function onKeydown(e) {
      if (e.keyCode === 32 && !e.repeat) {
        buzzButton.current.click();
        e.preventDefault();
      }
    }
    window.addEventListener('keydown', onKeydown);
    return () => window.removeEventListener('keydown', onKeydown);
  }, []);

  const players = !game.gameMetadata
    ? []
    : game.gameMetadata
      .filter((p) => p.name)
      .map((p) => ({ ...p, id: String(p.id) }));
  // host is lowest active user
  const firstPlayer =
    get(
      sortBy(players, (p) => parseInt(p.id, 10)).filter((p) => p.connected),
      '0'
    ) || null;
  const isHost = get(firstPlayer, 'id') === game.playerID;

  const queue = sortBy(values(game.G.queue), ['timestamp']);
  const buzzedPlayers = queue
    .map((p) => {
      const player = players.find((player) => player.id === p.id);
      if (!player) {
        return {};
      }
      return {
        ...p,
        name: player.name,
        connected: player.connected,
      };
    })
    .filter((p) => p.name);
  // active players who haven't buzzed
  const activePlayers = orderBy(
    players.filter((p) => !some(queue, (q) => q.id === p.id)),
    ['connected', 'name'],
    ['desc', 'asc']
  );

  const timeDisplay = (delta) => {
    if (delta > 1000) {
      return `+${round(delta / 1000, 2)} s`;
    }
    return `+${delta} ms`;
  };

  const copyURL = () => {
    const url = window.location.href;
    navigator.clipboard.writeText(url).then(function () {
      alert("Invite Link Copied !");
    }).catch(function (error) {
      console.error('Unable to copy URL: ', error);
    });
  };

  return (
    <div>
      <Header
        auth={game.headerData}
        clearAuth={() =>
          game.headerData.setAuth({
            playerID: null,
            credentials: null,
            roomID: null,
          })
        }
        sound={sound}
        setSound={() => setSound(!sound)}
      />
      <Container>
        <section>
          <p id="room-title">Room {game.gameID}</p>
          <div className="button-container" >
            <button class="text-button" id="copyButton" style={{ color:
'#ff9999', fontWeight: 'lighter', border: 'none', background: 'none',
textDecoration: 'underline', cursor: 'pointer', fontSize: 'small' }}
onClick={copyURL}>Invite Link? Click here</button>
          </div>
          {!game.isConnected ? (
            <p className="warning">Disconnected - attempting to reconnect...</p>
          ) : null}
          <div id="buzzer">
            <button
              ref={buzzButton}
              disabled={buzzed || game.G.locked}
              onClick={() => {
                if (!buzzed && !game.G.locked) {
                  attemptBuzz();
                }
              }}
            >
              {game.G.locked ? 'Locked' : buzzed ? 'Buzzed' : 'Buzz'}
            </button>
          </div>
          {isHost ? (
            <div className="settings">
              <div className="button-container">
                <button
                  className="text-button"
                  onClick={() => game.moves.toggleLock()}
                >
                  {game.G.locked ? 'Unlock buzzers' : 'Lock buzzers'}
                </button>
              </div>
              <div className="button-container">
                <button
                  disabled={isEmpty(game.G.queue)}
                  onClick={() => game.moves.resetBuzzers()}
                >
                  Reset all buzzers
                </button>
              </div>

              <div className="divider" />
            </div>
          ) : null}
        </section>
        <div className="queue">

          <p>Players Buzzed</p>
          <ul>
            {buzzedPlayers.map(({ id, name, timestamp, connected }, i) => (
              <li key={id} className={isHost ? 'resettable' : null}>
                <div
                  className="player-sign"
                  onClick={() => {
                    if (isHost) {
                      game.moves.resetBuzzer(id);
                    }
                  }}
                >
                  <div className={`name ${!connected ? 'dim' : ''}`}>
                    {name}
                    {!connected ? (
                      <AiOutlineDisconnect className="disconnected" />
                    ) : (
                      ''
                    )}
                  </div>
                  {i > 0 ? (
                    <div className="mini">
                      {timeDisplay(timestamp - queue[0].timestamp)}
                    </div>
                  ) : null}
                </div>
              </li>
            ))}
          </ul>
        </div>
        <div className="queue">
          <p>Players Joined</p>
          <ul>
            {activePlayers.map(({ id, name, connected }) => (
              <li key={id}>
                <div className={`name ${!connected ? 'dim' : ''}`}>
                  {name}
                  {!connected ? (
                    <AiOutlineDisconnect className="disconnected" />
                  ) : (
                    ''
                  )}
                </div>
              </li>
            ))}
          </ul>
        </div>
      </Container>
    </div>
  );
}
```

```
import React from 'react';
import { useParams } from 'react-router-dom';
import { Container, Spinner } from 'react-bootstrap';
import { Client } from 'boardgame.io/react';
import { SocketIO } from 'boardgame.io/multiplayer';
import { Buzzer } from '../lib/store';
import { GAME_SERVER } from '../lib/endpoints';
import Table from '../components/Table';
import Header from '../components/Header';

export default function Game({ auth, setAuth }) {
  const { id: roomID } = useParams();

  const loadingComponent = () => (
    <div>
      <Header
        auth={auth}
        clearAuth={() =>
          setAuth({
            playerID: null,
            credentials: null,
            roomID: null,
          })
        }
      />
      <Container className="container-loading">
        <Spinner animation="border" role="status">
          <span className="sr-only">Loading...</span>
        </Spinner>
      </Container>
    </div>
  );

  const App = Client({
    game: Buzzer,
    board: Table,
    multiplayer: SocketIO({ server: GAME_SERVER }),
    debug: false,
    loading: loadingComponent,
  });

  return (
    <main id="game">
      <div className="primary">
        <App
          gameID={roomID}
          playerID={String(auth.playerID)}
          credentials={auth.credentials}
          headerData={{ ...auth, setAuth }}
        />
      </div>
    </main>
  );
}
```

FULL CODE : https://github.com/abhimanyurajeesh/BuzzHub

4.SYSTEM CONFIGURATION

The system configuration for the BuzzHub - Online Buzzer System would include both hardware and software requirements to ensure its proper functioning. Below are the hardware and software requirements for building this project:

Hardware Requirements:

1. **Server:**
    - The system would require a server to host the web application and manage the backend operations.
    - The server should have sufficient processing power, memory (RAM), and storage to handle concurrent user connections and data processing tasks efficiently.
    - Recommended specifications would include multi-core processors, at least 4GB of RAM (though more may be beneficial for scalability), and ample storage space for storing application files and database data.
2. **Network Infrastructure:**
    - A reliable network infrastructure is necessary to ensure seamless communication between the server and clients accessing the web application.
    - This includes high-speed internet connectivity and proper network configuration to handle incoming and outgoing traffic effectively.
3. **Client Devices:**
    - Users would access the BuzzHub application through various client devices, including desktop computers, laptops, tablets, and smartphones.
    - These devices should have compatible web browsers to access the web-based application.

Software Requirements:

1. **Operating System:**

- The server hosting the BuzzHub application can run on various operating systems, including Linux distributions (e.g., Ubuntu, CentOS), Windows Server, or others.
- The choice of operating system would depend on the preferences and expertise of the system administrators.

2. **Web Server:**
    - The system would require a web server software to serve the web application to clients over the internet.
    - Popular choices for web servers include Apache HTTP Server, Nginx, Microsoft Internet Information Services (IIS), etc.
3. **Database Management System (DBMS):**
    - A relational database management system (RDBMS) is necessary to store and manage data related to users, game sessions, questions, answers, etc.
    - Commonly used RDBMS options include MySQL, PostgreSQL, Microsoft SQL Server, or SQLite.
4. **Programming Languages and Frameworks:**
    - The BuzzHub application can be developed using various programming languages and frameworks for both frontend and backend development.
    - Frontend technologies may include HTML, CSS, JavaScript, and frontend frameworks/libraries like React.js, Angular, or Vue.js.
    - Backend technologies may include server-side scripting languages like Node.js, Python, PHP, along with frameworks like Express.js, Django, Flask, etc.
5. **Other Software Dependencies:**
    - The system may have dependencies on additional software libraries, modules, or packages required for specific functionalities such as real-time communication, audio processing, authentication, etc.

- These dependencies would vary based on the features and requirements of the BuzzHub application.

4.1 HARDWARE REQUIREMENTS

RAM	4.00GB
HARD DISK	SSD or HDD
PROCESSOR	Intel(R) Core(TM) i3 7th Gen
PROCESSING SPEED	1.60GHz - 2.30 GHz

4.2 SOFTWARE REQUIREMENTS

FRONT END	HTML, CSS, React
BACK END	Node.js, RESTful API
TOOLS	MICROSOFT EXCEL 365, ADOBE ILLUSTRATOR
OPERATING SYSTEM	WINDOWS 10
DOCUMENTATION	MICROSOFT WORD 365

5. DETAILS OF SOFTWARE

5.1 OVERVIEW OF FRONTEND

HTML (HyperText Markup Language) is a markup language used to create the structure and content of web pages. It forms the foundation of the front-end for most web applications, including BuzzHub, an online buzzer system for multiplayer quiz bowl games.

The HTML front-end of BuzzHub comprises a series of HTML pages defining the structure and content of the user interface. These pages contain elements such as headings, paragraphs, lists, tables, forms, and other components to display data and receive user input.

Additionally, the HTML front-end utilizes cascading style sheets (CSS) to define the visual style of the user interface. CSS is employed to specify colors, fonts, spacing, and layout, enhancing the aesthetic appeal and usability of BuzzHub.

Furthermore, BuzzHub's front-end incorporates React.js, a JavaScript library for building user interfaces. React.js enables the creation of dynamic, interactive components within the HTML pages, enhancing user experience and facilitating seamless real-time interaction during quiz bowl games.

Overall, the HTML front-end of a timetable management system would be designed to provide a user-friendly, responsive, and visually appealing interface that allows users to view and interact with scheduling information in an efficient and convenient way.

FEATURES

1. Structural Elements: HTML provides a set of structural elements that allow developers to create the basic structure of a web page, including headings, paragraphs, lists, tables, and forms.
2. Semantic Elements: HTML also provides a set of semantic elements that describe the content of the web page in a meaningful way, such as headers, footers, main content areas, and navigation menus.
3. Cross-Browser Compatibility: HTML, CSS, and React.js ensure consistent performance across different web browsers and devices.

4. Accessibility: HTML provides built-in support for creating accessible web pages that can be accessed and used by people with disabilities, such as screen readers and keyboard navigation.
5. Extensibility: HTML is extensible, meaning that developers can define their own custom elements and attributes to create more specialized markup for their specific needs.
6. Embeddability: HTML provides a way to embed different types of media and content within web pages, such as images, videos, audio files, and interactive widgets.
7. Compatibility with Other Technologies: HTML can be used in conjunction with other web technologies such as CSS and JavaScript to create dynamic, interactive web pages with rich functionality and visual appeal.

DISADVANTAGES

1. Limited Styling Options: HTML is primarily a markup language for defining the structure and content of a web page, and it has limited options for styling the page. To style a web page, developers typically use CSS (Cascading Style Sheets), which is a separate technology that works in conjunction with HTML.

2. Complex Layouts: HTML can be challenging to use for complex page layouts, particularly when the layout involves a mix of text, images, and other media. Developers often need to use CSS and JavaScript to create complex layouts, which can add complexity to the development process.

3. Browser Compatibility Issues: While HTML is designed to work on all modern web browsers, there can still be compatibility issues between different browsers, particularly with older versions of browsers. Developers need to test their HTML code on multiple browsers to ensure that it works correctly.

4. Security Vulnerabilities: HTML can be vulnerable to security attacks such as cross-site scripting (XSS) and SQL injection. Developers need to take care to ensure that their HTML code is secure and that they follow best practices for web security.

5. Limited Interactivity: While HTML can be used to create interactive web pages, it has limited options for creating advanced interactivity such as animations and complex user

interfaces. Developers often need to use JavaScript and other technologies to create more advanced interactivity on web pages.

THE SOFTWARE INCLUDES

- HTML, CSS, and React.js as a front end
- Node.js as a back end

5.2 OVERVIEW OF BACKEND

The backend of BuzzHub, an online buzzer system for multiplayer quiz bowl games, serves as the central component responsible for handling server-side logic, data management, and communication with the front-end. It plays a crucial role in ensuring the smooth operation and functionality of the application.

Key components and features of the backend in BuzzHub include:

1. **Node.js Environment:** BuzzHub leverages Node.js, a JavaScript runtime environment, for building scalable and efficient server-side applications. Node.js enables non-blocking, event-driven architecture, making it suitable for handling concurrent connections and real-time interactions in multiplayer gaming environments.

2. **Express.js Framework:** Express.js, a web application framework for Node.js, is utilized in BuzzHub for building robust and flexible server-side APIs and routes. Express.js simplifies the process of handling HTTP requests, routing, middleware integration, and error handling, facilitating the development of a modular and maintainable backend architecture.

3. **WebSocket Protocol:** WebSocket protocol is employed in BuzzHub to establish persistent, full-duplex communication channels between the server and connected clients. WebSocket enables real-time data exchange,

allowing players to buzz in, receive game updates, and interact with each other instantaneously during quiz bowl sessions (socket.io).

4. **Database Management:** BuzzHub employs a memory-based storage mechanism using the Map object to manage game-related data, user profiles, game sessions, and leaderboard information. While it doesn't utilize a traditional database management system like MongoDB or MySQL, the backend efficiently handles data storage and retrieval operations, including CRUD operations and data validation, ensuring data integrity and optimal performance within the memory storage.

5. **Authentication and Authorization:** Authentication and authorization mechanisms are implemented in the backend to secure access to BuzzHub's features and resources. User authentication is typically based on JWT (JSON Web Tokens) or session-based authentication, while authorization rules define user roles and permissions for accessing specific endpoints and functionalities.

6. **Real-time Game Logic:** The backend of BuzzHub contains game logic components responsible for managing game sessions, enforcing game rules, and handling player interactions in real-time. This includes functionalities such as processing buzzer events, tracking scores, validating responses, and determining game outcomes based on quiz bowl rules.

7. **Scalability and Performance:** The backend architecture of BuzzHub is designed to be scalable and performant, capable of handling a large number of concurrent connections and game sessions. Techniques such as load balancing, caching, and horizontal scaling may be employed to ensure optimal performance under varying traffic loads.

5.3 ABOUT THE PLATFORM

BuzzHub is built on a modern and robust tech stack, leveraging various technologies to provide a seamless and engaging user experience. At its core, the backend is powered by Node.js, a lightweight and efficient runtime environment for server-side applications. The backend logic is implemented using the Koa web framework, known for its simplicity and scalability in building web servers and APIs.

For real-time communication and game management, BuzzHub utilizes the boardgame.io library, which offers powerful abstractions for handling game state, player actions, and WebSocket-based communication. This allows for smooth and synchronized gameplay experiences across multiple users. Additionally, the platform integrates Socket.IO, a JavaScript library for real-time web applications, to facilitate bidirectional communication between the server and clients. Socket.IO enhances the responsiveness and interactivity of the platform, enabling instant updates and notifications during gameplay sessions.

On the frontend, BuzzHub leverages HTML, CSS, and React.js, a popular JavaScript library for building user interfaces. React.js enables the creation of dynamic and responsive UI components, ensuring an intuitive and engaging user interaction. Additionally, CSS is used to style the frontend components, providing a visually appealing interface that enhances the overall user experience.

By leveraging these technologies, BuzzHub delivers a modern and feature-rich online buzzer system that caters to the needs of trivia enthusiasts and gaming communities worldwide. The platform's architecture and design ensure scalability, performance, and maintainability, allowing for future enhancements and updates to meet evolving user requirements.

Socket.IO: Socket.IO is a JavaScript library that enables real-time, bidirectional communication between web clients and servers. It uses WebSocket technology to establish a persistent connection between the client and server, allowing for instant data exchange. Socket.IO simplifies the implementation of real-time features in web applications, such as chat systems, live updates, and multiplayer gaming. In BuzzHub, Socket.IO facilitates seamless communication between players and the server, enabling synchronized gameplay and interactive experiences.

boardgame.io: boardgame.io is a powerful game development library for creating turn-based and real-time multiplayer games. It provides a framework for managing game state, handling player actions, and synchronizing game logic between clients and servers. With boardgame.io, developers can focus on defining the rules and mechanics of their games while the library takes care of the underlying infrastructure. In BuzzHub, boardgame.io powers the game management system, ensuring smooth and engaging gameplay for participants.

Koa: Koa is a lightweight and modular web framework for Node.js, designed by the creators of Express.js. It offers a more expressive and elegant approach to building web servers and APIs by utilizing ES6 features such as async/await and generators. Koa middleware functions allow for fine-grained control over request handling and response generation, enabling developers to create efficient and scalable server applications. In BuzzHub, Koa serves as the backend framework, providing a solid foundation for implementing routing, middleware, and request handling logic.

React.js: React.js is a JavaScript library for building user interfaces, developed by Facebook. It facilitates the creation of interactive and reusable UI components by using a declarative and component-based approach. React.js efficiently updates the DOM in response to data changes, resulting in fast and responsive user interfaces. In BuzzHub, React.js powers the frontend interface, enabling the dynamic rendering of game elements, player interactions, and real-time updates. Its component-based architecture promotes code reusability and maintainability, making it an ideal choice for complex web applications like BuzzHub.

6. TESTINNG

Testing is an integral aspect of developing BuzzHub, and it's crucial to incorporate it early in the development process. By making testing a part of the requirement gathering phase, you ensure that the software meets the stakeholders' needs effectively. Throughout the software development lifecycle, feedback from users plays a significant role in shaping the future of the application. Thus, it's essential to consider the entire lifecycle and how user feedback will influence the application's evolution.

The tools and techniques utilized in BuzzHub development aid in responsiveness to changes without incurring additional costs. However, it's important to adopt new tools and process improvements gradually, evaluating the results at each step. Testing is not just a standalone phase but an integral part of the lifecycle. It involves identifying needs, developing code to fulfill them, and checking if stakeholders are satisfied with the outcome. This cycle of development, testing, and refinement may occur frequently, as in the case of rapid ice cream vending projects, or over longer periods for carefully specified systems like healthcare support systems.

In the context of BuzzHub, testing serves as a proxy for user satisfaction. Rather than relying on releasing the software and waiting for feedback, testing ensures that the product meets stakeholders' needs before reaching users. Tests are designed based on stakeholders' requirements and executed to validate the software's functionality. The principle is clear: tests represent requirements, and the development process isn't complete until the tests pass successfully. Therefore, tests should be derived from and linked to the requirements, serving as a means to ensure that the software meets stakeholders' expectations effectively.

. In this light, two important principles become clear:

- Tests represent requirements. Whether you write user stories on sticky notes on the wall, or use cases in a big thick document, your tests should be derived from and linked to those requirements. And as we've said, devising tests is a good vehicle for discussing the requirements.

- We're not done till the tests pass. The only useful measure of completion is when tests have been performed successfully.

SOFTWARE TESTING TYPES

BLACK BOX TESTING: Internal system design is not considered in this type of testing. Tests are based on requirements and functionality.

WHITE BOX TESTING: This testing is based on knowledge of the internal logic of an application's code. Also known as Glass box Testing. Internal software and code working should be known for this type of testing. Tests are based on coverage of code statements, branches, paths, conditions.

UNIT TESTING: Testing of individual software components or modules. Typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code. may require developing test driver modules or test harnesses.

INCREMENTAL INTEGRATION TESTING: Bottom-up approach for testing i.e., continuous testing of an application as new functionality is added; Application functionality and modules should be independent enough to test separately. done by programmers or by testers.

INTEGRATION TESTIN: Testing of integrated modules to verify combined functionality after integration. Modules are typically code modules, individual applications, client and server applications on a network, etc. This type of testing is especially relevant to client/server and distributed systems.

FUNCTIONAL TESTING: This type of testing ignores the internal parts and focus on the output is as per requirement or not. Black-box type testing geared to functional requirements of an application.

SYSTEM TESTING: Entire system is tested as per the requirements. Black-box type testing that is based on overall requirements specifications, covers all combined parts of a system.

END-TO-END TESTING: Similar to system testing, involves testing of a complete application environment in a situation that mimics real-world use, such as interacting with a database, using network communications, or interacting with other hardware, applications, or systems if appropriate.

SANITY TESTING: Testing to determine if a new software version is performing well enough to accept it for a major testing effort. If the application is crashing for initial use, then the system is not stable enough for further testing and build or application is assigned to fix.

REGRESSION TESTING: Testing the application for modification in any module or functionality. Difficult to cover all the systems in regression testing so typically automation tools are used for these testing types.

ACCEPTANCE TESTING: Normally this type of testing is done to verify if the system meets the member specified requirements. User or member do this testing to determine whether to accept application.

LOAD TESTING: It's a performance testing to check system behavior under load. Testing an application under heavy loads, such as testing of a web site under a range of loads to determine at what point the system's response time degrades or fails.

STRESS TESTING: System is stressed beyond its specifications to check how and when it fails. Performed under heavy load like putting large number beyond storage capacity, complex database queries, continuous input to system or database load.

PERFORMANCE TESTING: Term often used interchangeably with 'stress' and 'load' testing. To check whether the system meets performance requirements. Used different performance and load tools to do this.

USABILITY TESTING: User-friendliness check. Application flow is tested, can new user understand the application easily, Proper help documented whenever user stuck at any point. Basically, system navigation is checked in this testing.

INSTALL/UNINSTALL TESTING: Tested for full, partial, or upgrade install/uninstall processes on different operating systems under different hardware, software environment.

RECOVERY TESTIG: Testing how well a system recovers from crashes, hardware failures, or other catastrophic problems.

SECURITY TESTING: Can system be penetrated by any hacking way. Testing how well the system protects against unauthorized internal or external access. Checked if system, database is safe from external attacks.

COMPABILITY TESTING: Testing how well the software performs in a particular hardware/software/operating system/network environment and different combinations of the above.

COMPARISON TESTING: Comparison of product strengths and weaknesses with previous versions or other similar products.

ALPHA TESTING: In-house virtual user environment can be created for this type of testing. Testing is done at the end of development. Still minor design changes may be made because of such testing.

BETA TESTING: Testing typically done by end-users or others. Final testing before releasing application for commercial purpose.

NOTE: We have not performed all the tests

7. CONCLUSION AND FUTURE ENHANCEMENT

CONCLUSION

In conclusion, the development of BuzzHub represents a significant step towards revolutionizing the traditional buzzer system by harnessing the power of modern web technologies. With its intuitive online platform, real-time interaction capabilities, and scalable infrastructure, BuzzHub offers a dynamic and engaging experience for trivia enthusiasts worldwide.

FUTURE ENHANCEMENT

There is always room for some improvement. Looking ahead, BuzzHub has promising avenues for future enhancement and development to further solidify its position as a premier online buzzer system. Here are some potential areas for improvement and expansion:

- Enhanced User Experience: Continuously improving the user interface and experience to make navigation more intuitive and enjoyable for participants.

- Advanced Game Features: Introducing new game modes, customizable settings, and interactive elements to enhance gameplay and cater to a broader audience.

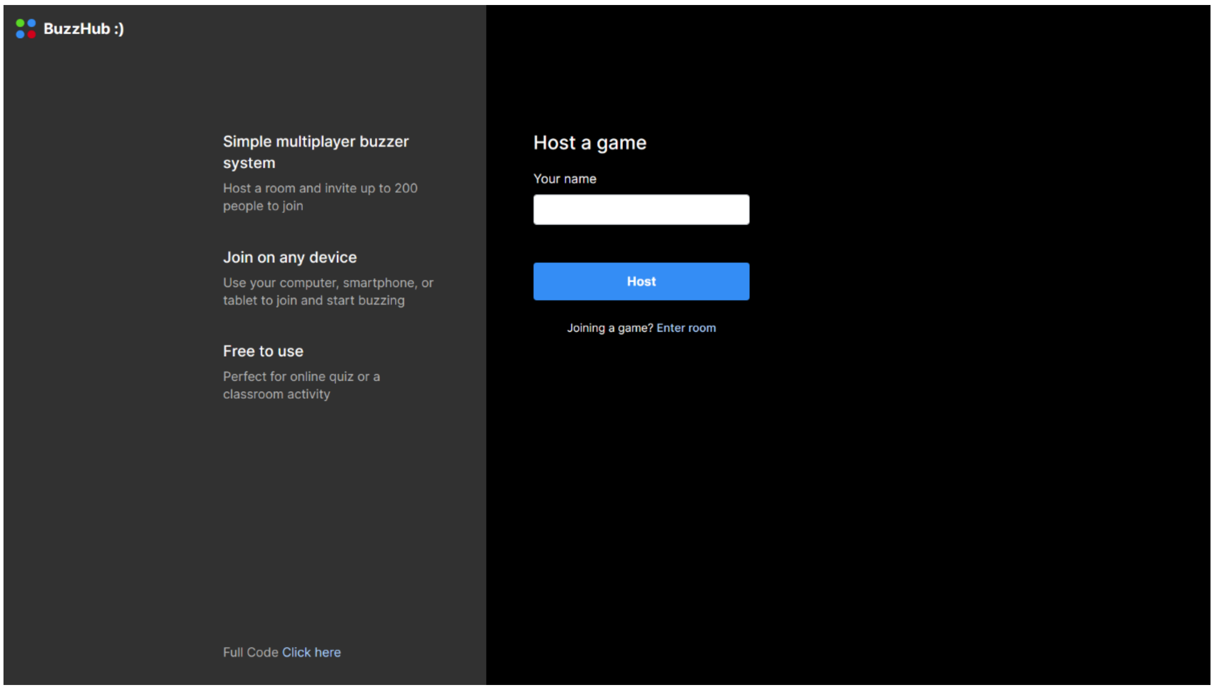
- Real-Time Communication: Chat features enable real-time communication between players during gameplay, allowing them to strategize, discuss answers, and engage in friendly banter. This enhances the overall gaming experience by fostering a sense of camaraderie and community among participants.

8. BIBLOGRAPHY

1. Node.js Foundation. "Node.js Documentation." Node.js, 2022, https://nodejs.org/en/docs/.
2. Express.js. "Express.js Guide." Express.js, 2022, https://expressjs.com/en/guide/routing.html.
3. Socket.IO. "Socket.IO Documentation." Socket.IO, 2022, https://socket.io/docs/v4/.
4. Boardgame.io. "Boardgame.io Guide." Boardgame.io, 2022, https://boardgame.io/documentation/#/.
5. MDN Web Docs. "WebSocket Tutorial." MDN Web Docs, 2022, https://developer.mozilla.org/en-US/docs/Web/API/WebSockets_API.
6. Flanagan, David. JavaScript: The Definitive Guide. O'Reilly Media, 2020
7. OpenAI. "GPT-3: Language Models are Few-Shot Learners. https://chat.openai.com/

9. APPENDICES A - SCREENSHOTS

HOST SIDE


BuzzHub :)

Simple multiplayer buzzer system
Host a room and invite up to 200 people to join

Join on any device
Use your computer, smartphone, or tablet to join and start buzzing

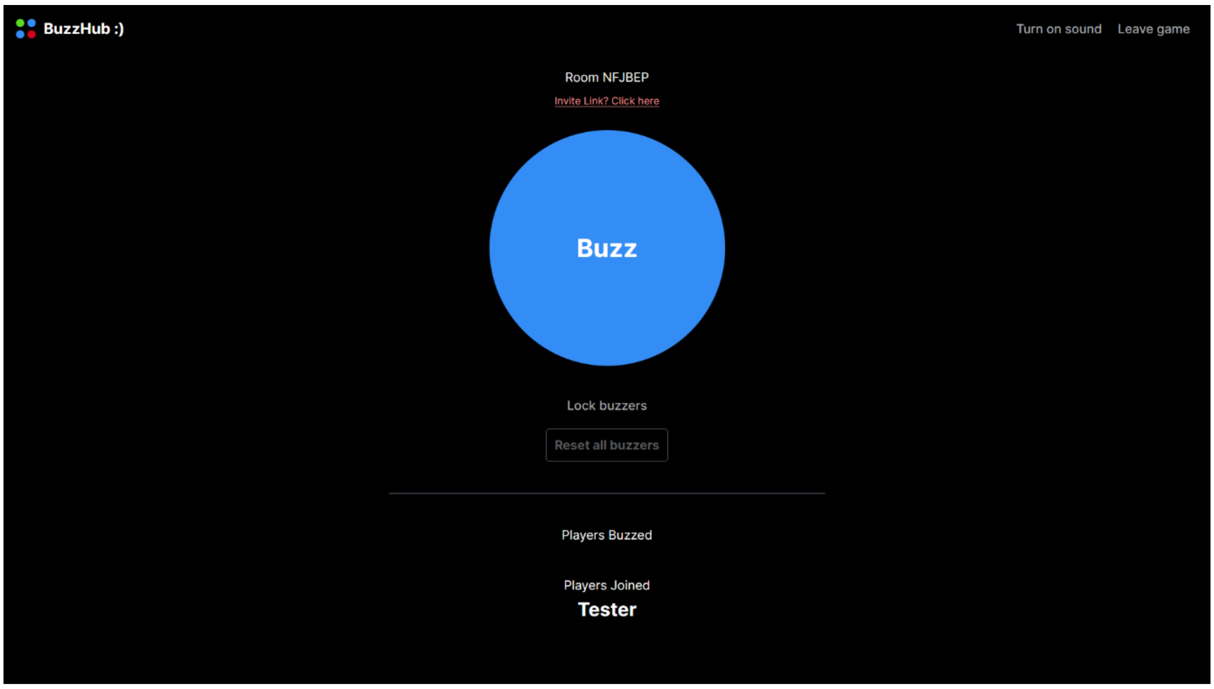
Free to use
Perfect for online quiz or a classroom activity

Full Code Click here

Host a game
Your name

Host

Joining a game? Enter room


BuzzHub :)

Turn on sound Leave game

Room NFJBEP
Invite Link? Click here

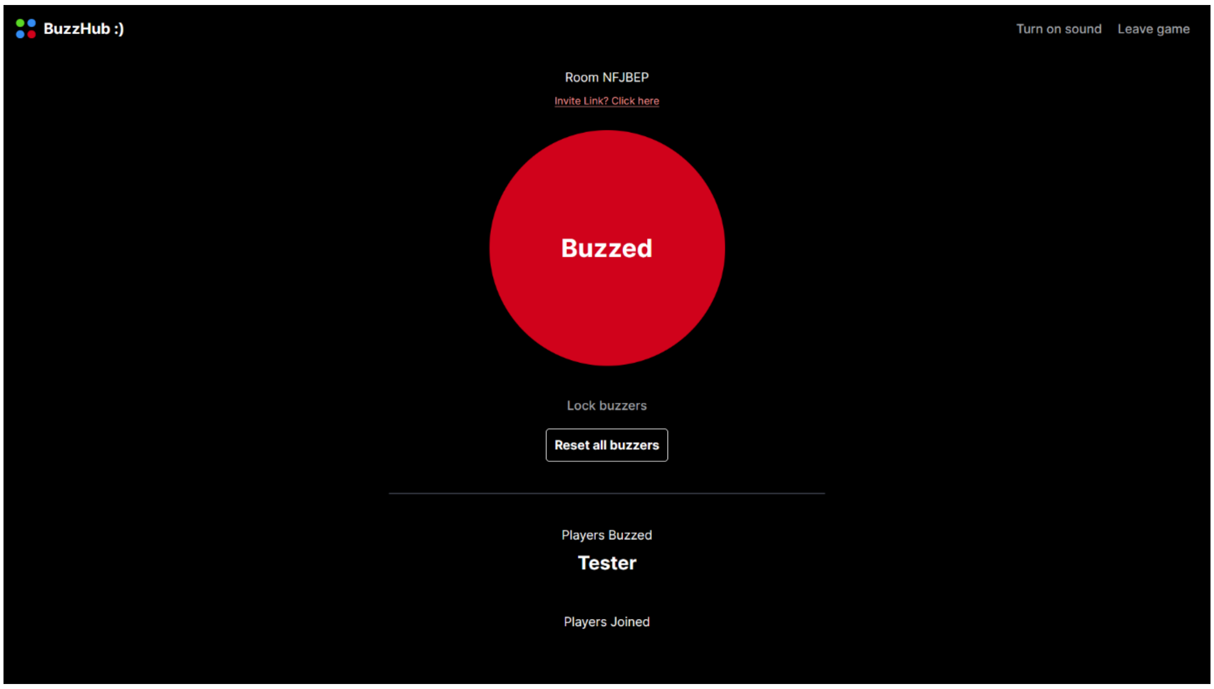
Buzz

Lock buzzers

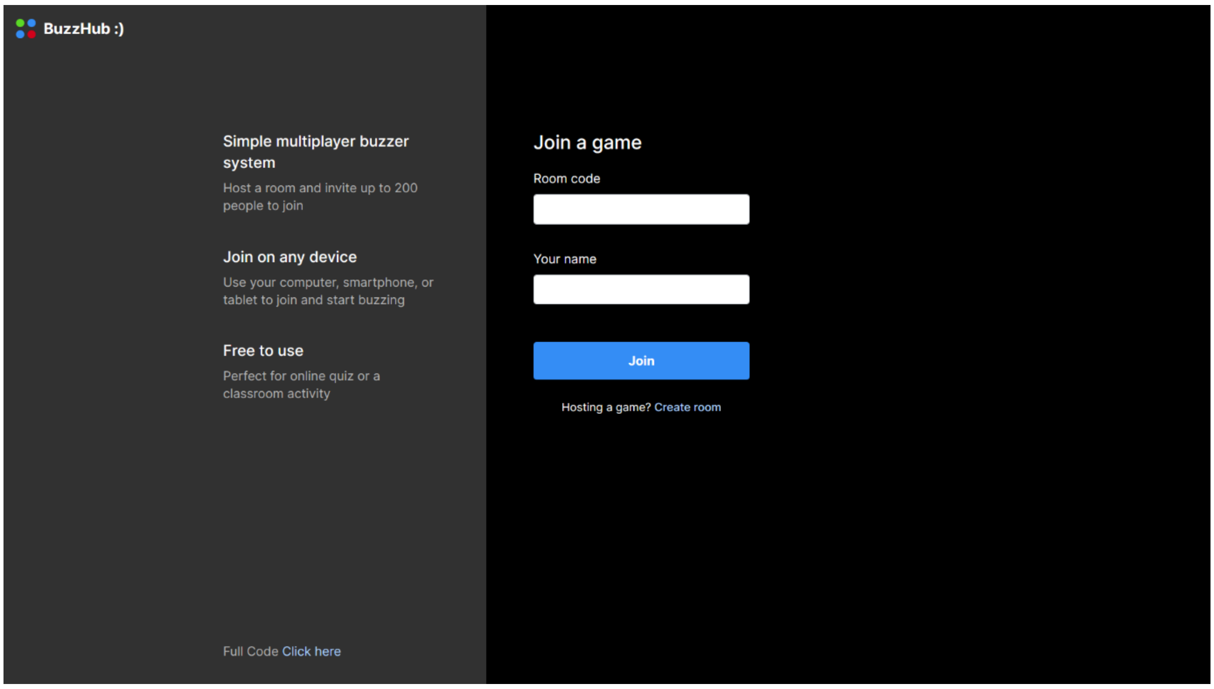
Reset all buzzers

Players Buzzed

Players Joined
Tester



PLAYER SIDE



Simple multiplayer buzzer system

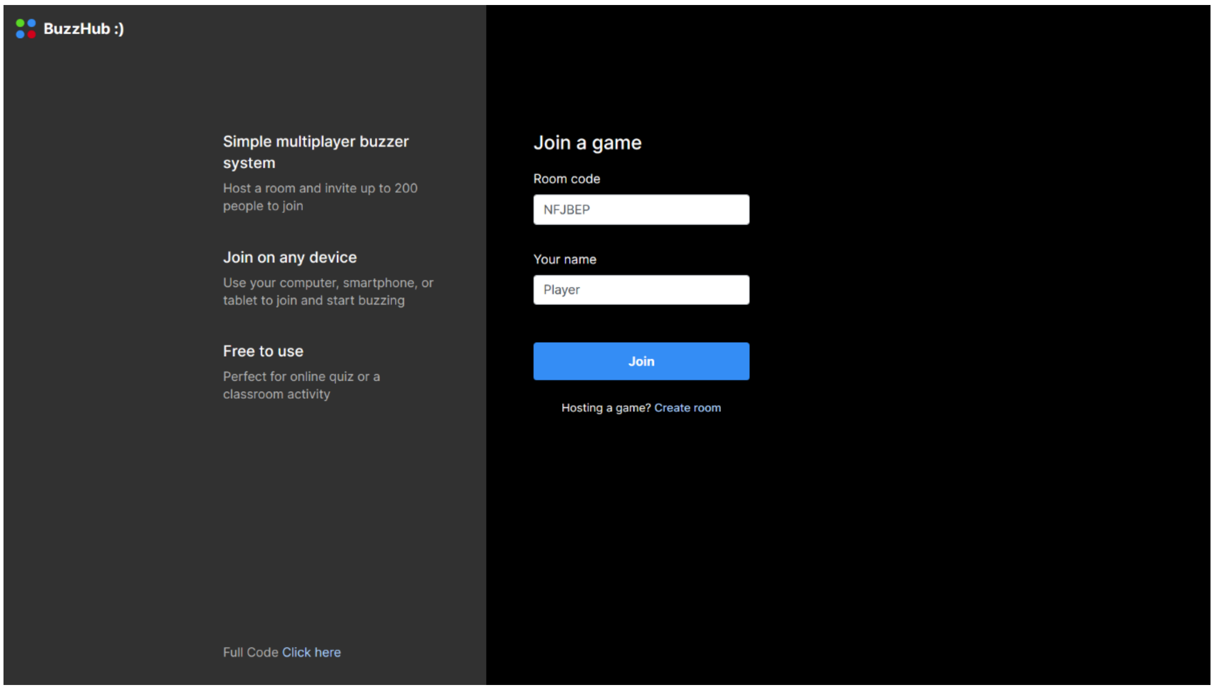
Host a room and invite up to 200 people to join

Join on any device

Use your computer, smartphone, or tablet to join and start buzzing

Free to use

Perfect for online quiz or a classroom activity


BuzzHub :)

Simple multiplayer buzzer system
Host a room and invite up to 200 people to join

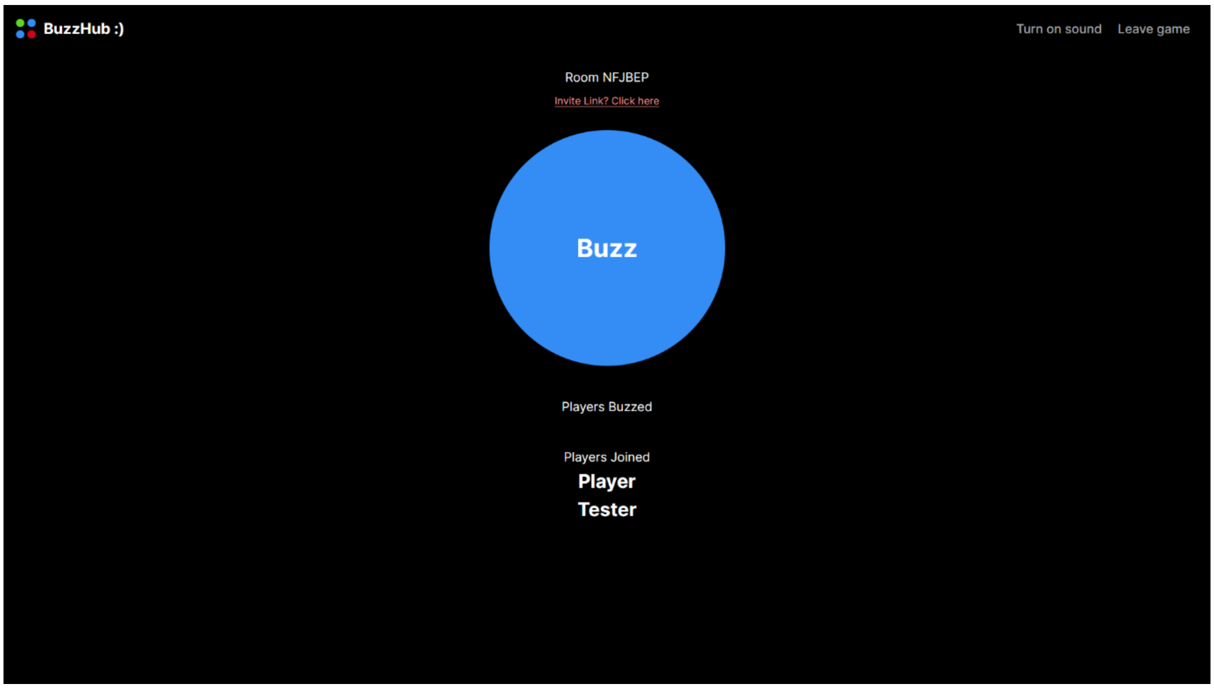
Join on any device
Use your computer, smartphone, or tablet to join and start buzzing

Free to use
Perfect for online quiz or a classroom activity

Full Code Click here

Join a game
Room code: NFJBEP
Your name: Player

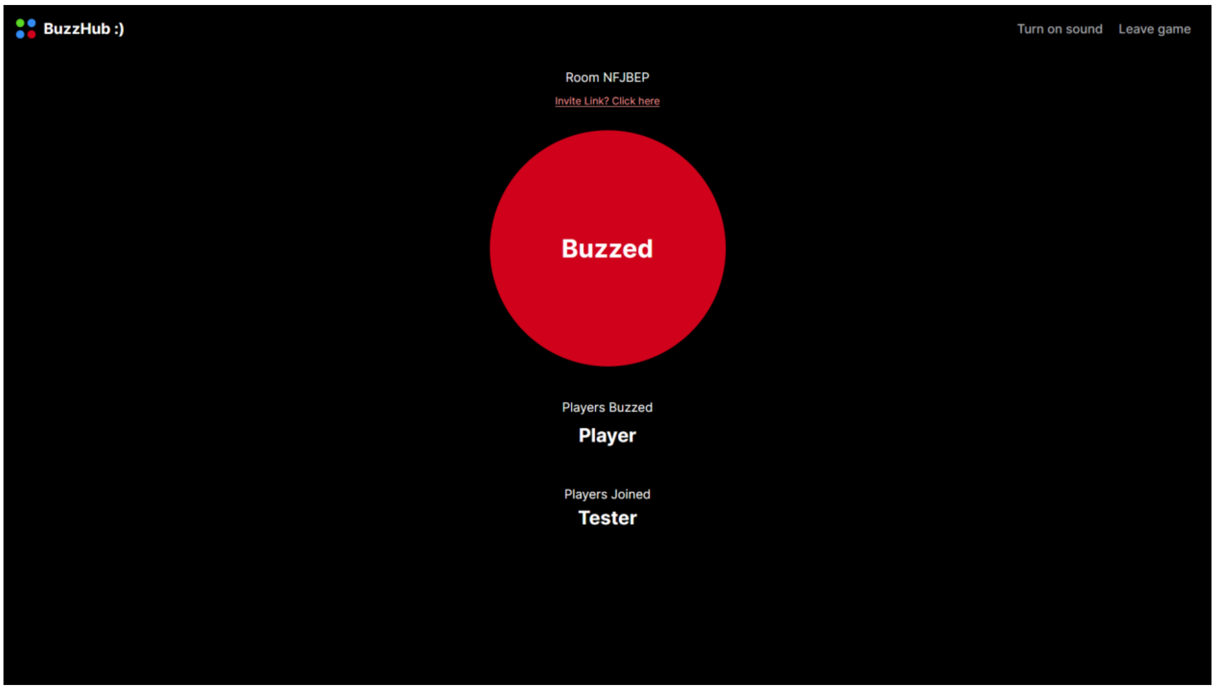
Hosting a game? Create room


BuzzHub :)

Room NFJBEP
Invite Link? Click here

Players Buzzed

Players Joined
Player
Tester


BuzzHub :)
Turn on sound
Leave game
Room NFJBEP
Invite Link? Click here
Buzzed
Players Buzzed
Player
Players Joined
Tester

10. APPENDICES B - SAMPLE REPORT OF TEST CASES

Test Case ID	Description	Test Steps	Expected Result	Actual Result	Pass/Fail
TC-HR-001	Host creating a room	1. Navigate to the host dashboard.	Host dashboard is displayed.	Host dashboard is displayed	Pass
		2. Click on the "Create Room" button.	Room creation form is displayed.	Room creation form is displayed.	Pass
		3. Enter valid details for room creation.	Room details are entered successfully.	Room details are entered successfully.	Pass
		4. Click on the "Host" button.	Room is successfully created.	Room is successfully created.	Pass
		5. Verify success message for room creation.	Message confirms successful room.	Opens the buzzer window	Pass
TC-PJ-002	Player joining a room	1. Navigate to the player dashboard.	Player dashboard is displayed.	Player dashboard is displayed.	Pass
		2. Enter the room ID provided by the host.	Room ID is entered.	Room ID is entered.	Pass
		3. Click on the "Join" button.	Player is successfully added to the room.	Player is successfully added to the room.	Pass

		4. Verify successful room joining message.	Message confirms successful joining of the room.	Opens the buzzer window	Pass
TC-PAJ-003	Another player attempting to join with same name	1. Navigate to the player dashboard.	Player dashboard is displayed.	Player dashboard is displayed.	Pass
		2. Enter the same room ID provided earlier.	Same room ID is entered.	Same room ID is entered.	Pass
		3. Enter the same player name as an already joined player.	Same player name is entered.	Same player name is entered.	Pass
		4. Click on the "Join" button.	Player is not allowed to join and error message is displayed.	Player is not allowed to join and message is displayed.	Pass
		5. Verify error message for duplicate player name.	Error message indicates that the player name is already in use.	Player name already taken.	Pass
TC-CE-004	Check if the room exists or not	1. Navigate to the player dashboard.	Player dashboard is displayed.	Player dashboard is displayed.	Pass
		2. Enter a non-existent room ID.	Non-existent room ID is entered.	Non-existent room ID is entered	Pass
		3. Click on the "Join" button.	Player is informed that the room does not exist.	Player is informed that the room does not exist.	Pass

		4. Verify error message for non-existent room.	Error message indicates that the room does not exist.	Unable to join room with this code	Pass
TC-FV-005	Field validation for blank inputs	1. Navigate to the host or player dashboard.	Dashboard is displayed.	Dashboard is displayed.	Pass
		2. Leave any mandatory field blank.	Mandatory fields are left blank.	Mandatory fields are left blank.	Pass
		3. Attempt to perform the respective action (e.g., create room, join room).	Action should not be allowed and error message should be displayed.	Action not allowed and error message displayed.	Pass
		4. Verify error messages for blank fields.	Error messages indicate that mandatory fields are required.	Please enter a room code/ Please enter your player name	Pass