

Table of Contents

CRC Cards	3
Software Architecture Diagram	7
3	
Three-Tier Architecture	7

CRC Cards

Class: Friends List Leaderboard AccountSettings Authenticator Reponsibility Reponsibility Reponsibility Reponsibility Store the names of the top 50 site users Change user's username Log users in (check submitted Contains list of users credentials) Keep track of whether the user based on XP and level Change user email Change user password Change user birthday Set profile preferences in the profile object Store the XP and level of each of those is logged in or not 50 users Log users out Log and report suspicious Set user's notification settings acitivity (multiple failed password (email subscriptions, reminders) attempts) Lock user accounts that might be under attack or act suspiciously Collaborators Collaborators Collaborators Collaborators Profile

DailyChallenge
Parent Class: Game

Reponsibility

Store the name of the daily challenge

Pull game description and assets from the corresponding class

Store XP yield Store obtainable badges

Collaborators

The game class corresponding to today's daily challenge (changes everyday)

Dashboard

Reponsibility

Store the name of the daily challenge Pull profile information from Profile object Pull top 50 users from Leaderboard

object Update the last played game (continue playing)

List a collection of games sorted into various categories

Collaborators

DailyChallenge Profile Leaderboard Game and all of its subclasses Profile

Reponsibility

Store user's badges Store user level and total XP Store user leaderboard ranking Pull user's username from AccountSettings object Pull user's profile picture from AccountSettings object

Add and store friends list

Collaborators

AccountSettings

-

Game (Abstract) Subclasses: DailyChallenge, StackGame, QueueGame, TreeTraversalGame, DictionaryGame, ArrayGame, IfStatementGame, HelloWorldGame

\mathbf{D}	nor	101	hil	1111
\neg	por	.51	OH	HV
	\sim .	_	~ 11	

Check game state (currently playing, loss, win)

Update user's XP count and level on game completion in Profile object

Update user's badge collection on achievement completion in Profile object

Validate user input

Update class attributes to reflect user's action in game Save Game state (if user leaves, doesn't have to start over)

Collaborators

Profile

QueueGame Parent class: Game	If Statement Game Parent class: Game	DictionaryGame Parent class: Game	
Reponsibility	Reponsibility	Reponsibility	
Update XP yield based on user's score	Update XP yield based on user's scor Store obtainable badges	Store XP yield Store obtainable badges	
Store obtaininable badges	Record the if conditions that the user	submitted Check whether the user answere the question correctly	
Update user's score after each marble action	Check that the user submitted valid if conditions through the dropdown lis		
Keep track of the order of marbles	Move the robot north, south, east, we depending on the if conditions specifi	st answered all questions	
Update game state to "won" when the amarbles are in the desired order	by the user Update the game state to "won" wher	Collaborators	
Record the user's action history (what moves have been made)	has reached the goal tile Calculate the user's score based on t number of moves the robot took to re the goal tile		
Collaborators	the goartile		
	Collaborators		
☐ TreeTraversalGame Parent class: Game	ArrayGame Parent class: Game	HelloWorldGame Parent class: Game	
Reponsibility	Reponsibility	Reponsibility	
Store XP yield Store obtainable badges Check if a user's traversal	Store XP yield Store obtainable badges	Store XP yield Store obtainable badges Check if the user has submitted their name through the text box	
choice alerts the criminal Update the game state to "won" if the user has chooses a	Check if the user dragged the shape to the correct array index		
traversal order that doesn't	Update the game state to "won"	Check that the user has submitted	

Update the game state to "won" when all questions have been answered correctly

Collaborators

alert the criminal

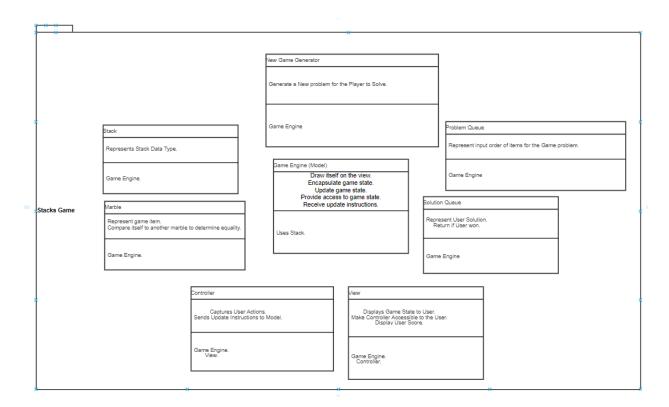
Validate user input

Collaborators

"Hello World" through the text box

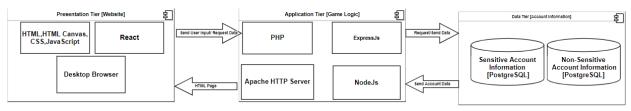
Update the game state to "won" when "Hello World" has been printed to the screen

Collaborators



Software Architecture Diagram

Three-Tier Architecture



The three-tier architecture diagram is responsible for describing how our system is divided into components. Our system is divided into 3 tiers, those being the Presentation Tier (website), the Application Tier (web server), and the Data Tier (database).

The Presentation Tier contains the website layer. The website layer represents our website as it is displayed to user. The website layer includes all HTML, CSS, and JavaScript files using the React library. The purpose of the website layer is to take user input and send it to the Application Tier for processing and validation. This includes usernames, emails, passwords, changes to other account information, and game inputs. The website layer is also responsible for requesting HTTP pages from the web server and displaying the HTML, CSS, and JavaScript that it receives from the Application Tier.

The Application Tier contains the web server layer. The web server layer includes the Apache HTTP Web Server that hosts our website and all PHP files and JavaScript files using the ExpressJs framework. The web server layer also includes the NodeJs environment required to execute the JavaScript files. The purpose of the web server layer is to process and validate all user input received from the website layer (Presentation Tier). The web server layer is responsible for handling all game logic. Therefore, all game inputs are processed, validated, and reflected in this layer. Additionally, the web server layer is responsible for querying and updating account data (e.g. usernames, emails, passwords) on the PostgreSQL databases in the Data Tier to perform password authentication and update user account information. Queries the database layer are made through prepared statements to avoid SQL injection attacks.

Within the Data Tier lives the database layer. The database layer includes the PostgreSQL databases responsible for storing usernames, passwords, emails, and other account information. The purpose of the database layer is to send non-sensitive

account information to the web server layer (Application Tier) for processing and to send sensitive account information to the web server layer for password authentication. The database layer is also responsible for reflecting any changes to user account information in the corresponding database.

The following is an example use case of our system. A user logs into our website with a correct username and password. The HTML file in our website layer sends the username and password to a PHP file in our web server layer by way of HTML form and POST request. The PHP file in our web server layer queries the row in the PostgreSQL database (sensitive account information) with the given username and password with a prepared statement. The database fulfills the query request by sending the row with matching username and password to the PHP file. The PHP file verifies that the query returned a non-empty output and then has the Apache HTTP Server send the "logged-in" HTML page to the website layer. The website layer displays the "logged-in" HTML page to the user. The user has now securely logged into our website.

If the user inputs an invalid password, the query to the PostgreSQL database returns an empty output. The PHP file would then log the failed password attempt and have the Apache HTTP server send an "error" HTML page to the website layer. After 3 failed password attempts, the user will be locked out of their account 5 minutes. The lockout times doubles with each subsequent failed attempt.