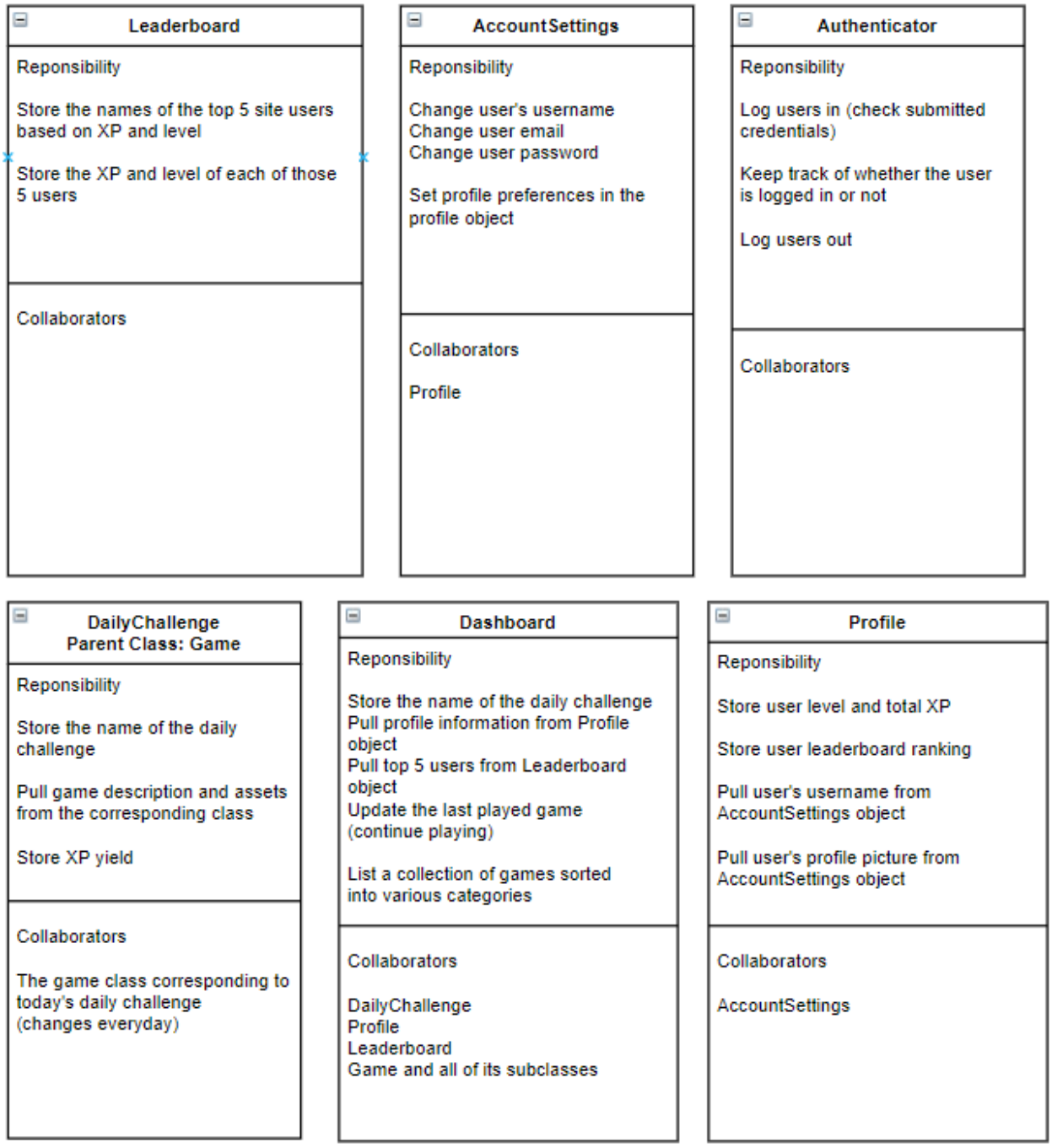
DreamTeam System Design Document

Table of Contents

CRC Cards


Leaderboard
Reponsibility
Store the names of the top 5 site users based on XP and level
Store the XP and level of each of those 5 users
Collaborators
AccountSettings
Reponsibility
Change user's username
Change user email
Change user password
Set profile preferences in the profile object
Collaborators
Profile
Authenticator
Reponsibility
Log users in (check submitted credentials)
Keep track of whether the user is logged in or not
Log users out
Collaborators
DailyChallenge
Parent Class: Game
Reponsibility
Store the name of the daily challenge
Pull game description and assets from the corresponding class
Store XP yield
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 5 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 level and total XP
Store user leaderboard ranking
Pull user's username from AccountSettings object
Pull user's profile picture from AccountSettings object
Collaborators
AccountSettings

Game (Abstract) **Subclasses: DailyChallenge, StackGame, QueueGame, TreeTraversalGame, DictionaryGame, ListGame, IfStatementGame, HelloWorldGame**
Reponsibility Check game state (currently playing, loss, win) Update user's XP count and level on game 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
Reponsibility Store XP yield Update user's score after each marble action Keep track of the order of marbles Update game state to "won" when the marbles are in the desired order Record the user's action history (what moves have been made)
Collaborators

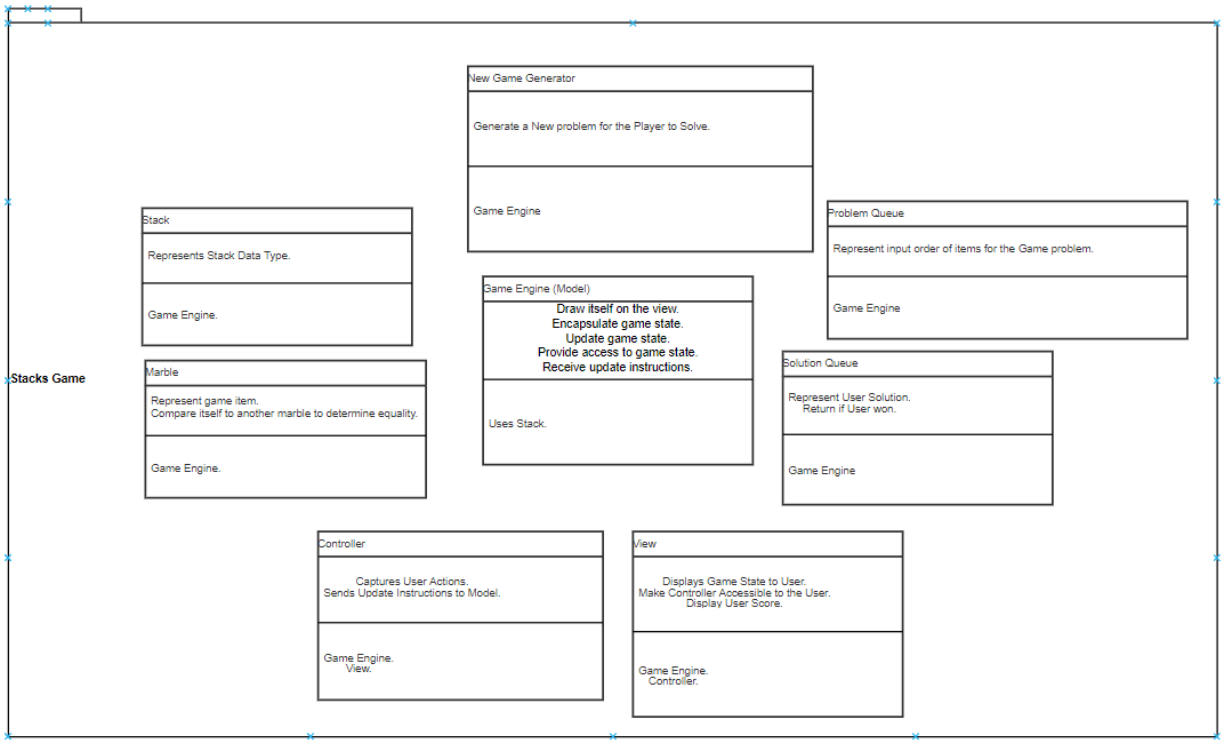
IfStatementGame Parent class: Game
Reponsibility Update XP yield based on user's score Record the if conditions that the user submitted Check that the user submitted valid if conditions through the dropdown list Move the robot north, south, east, west depending on the if conditions specified by the user Update the game state to "won" when the robot has reached the goal tile Calculate the user's score based on the number of moves the robot took to reach the goal tile
Collaborators

DictionaryGame Parent class: Game
Reponsibility Store XP yield Check whether the user's dictionary matches the target dictionary Add/Update key-value pairs in the user's dictionary Remove key-value pairs in the user's dictionary Update game state to "won" when the user's dictionary matches the target dictionary
Collaborators

TreeTraversalGame Parent class: Game
Reponsibility Store XP yield Check that a user has selected the code snippet that builds a tree such that the given tree traversal order would return the specified values Increment the user's score when a correct code snippet has been chosen Decrement the user's score when an incorrect code snippet has been chosen Validate user input
Collaborators

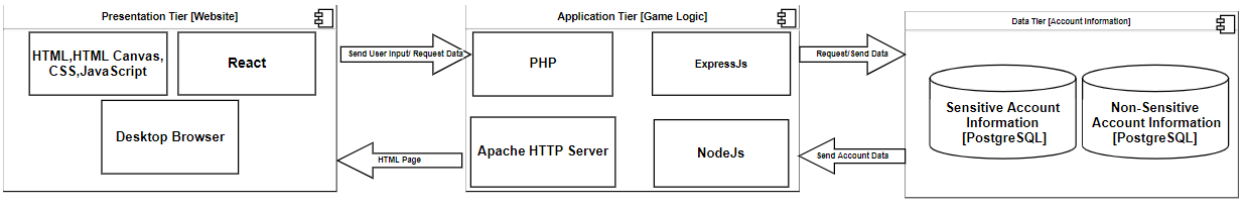
ListGame Parent class: Game
Reponsibility Store XP yield Generate and display shopping items on the screen Remove an item at a specific index from the shopping cart Append an item to the shopping cart Check if an item in the shopping cart matches any item on the shopping list Update the game state to "won" when the correct shopping items have been dragged to the cart Move a shopping item in the direction of the cursor when dragged by the user Sort items in the shopping list by name
Collaborators

HelloWorldGame Parent class: Game
Reponsibility Store XP yield Check if the user has submitted their name through the text box Check that the user has submitted "Hello World" through the text box Update the game state to "won" when "Hello World" has been printed to the screen
Collaborators


Stacks Game
New Game Generator
Generate a New problem for the Player to Solve.
Game Engine
Stack
Represents Stack Data Type.
Game Engine.
Problem Queue
Represent input order of items for the Game problem.
Game Engine
Game Engine (Model)
Draw itself on the view.
Encapsulate game state.
Update game state.
Provide access to game state.
Receive update instructions.
Uses Stack.
Marble
Represent game item.
Compare itself to another marble to determine equality.
Game Engine.
Solution Queue
Represent User Solution.
Return if User won.
Game Engine
Controller
Captures User Actions.
Sends Update Instructions to Model.
Game Engine.
View.
View
Displays Game State to User.
Make Controller Accessible to the User.
Display User Score.
Game Engine.
Controller.

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 React components. 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, our Node.js API, and all PHP 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 React component on our website layer sends the username and password to our Node.js API on our web server layer by way of HTML form and GET request. The API on 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 API. The API sends the output to the React component. The React component verifies that the output is non-empty and 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 React component would then log the failed password attempt and display the corresponding error message to the user.