06

CIS045-3 Distributed Service Architectures – AY 24/25

Student Name: Warnakulasuriya Mudiyanselage Ihalagedara Dilshani Chamodya Balasuriya.

Student Number: 2424576

Student Signature: [illegible] **Date:17th November 2024**

Software Architecture (Low Coupling / High Cohesion)

How is your code structured? (e.g., components, files, libraries, classes, packages …)

The game's software architecture is Model-view-Controller (MVC), where models are defined in models, views are defined in react frontend components, controllers are defined in middleware and routes, and the technology used is MERN. The Frontend language used to develop the game is React, which follows a modular architecture allowing reusability and maintainability. Components are grouped based on their functionality, ensuring low coupling and high cohesion.

The main components include:

- Using isolated and dedicated components for specific tasks:
  1. Authentication Component: Login.js and Register.js manage user authentication.
  2. Game component: Components like Card.js, CardItem.js, and Game.js focus entirely on displaying and managing flipping cards.
- Centralized State Management: AuthContext and GameContext encapsulated all related state management logic for user authentication and game data. And also manage API calls.
- Utility function SetAuthToken.js is focused on setting the authentication token in the request header.

Event-Driven Architectures

What in your code triggers events (e.g., GUI, buttons, timeout …)?

Button Clicks:

- Game card flip: when a user clicks on a card, the game triggers the card-flip action. The card state changes from face-down to face-up, revealing the picture.
- Level Selection button: The user can select three different levels each having a different no of moves and cards
- Play Game button: Directing to Levels page.
- Submit button: The submit button allows the user to submit the answer to the banana game which fetches from API and compares the answer.
- Back button: Directing back to the previous page

Game over Condition: If the user fails to complete the game develop with the given no of moves, an event triggers and displays the game over screen with a random math equation dynamically load using Banana API.

Game processing events:

- Card Matching: When the user flips two cards, the game compares the two cards. If they match an event trigger to move state to display the card content.

- Game state update: The game state is updated periodically. These updates trigger UI changes, reflecting the no of remaining moves displaying how many games the user has won in each level.

Banana Game Trigger:

- If a player exceeds the maximum number of moves in the memory game, game over screen is displayed with the math question fetch using the Banana API.

User Authentication Events:

- Login: The login event triggers an API call to authenticate the user, and allow the user to access the protected routes and game data.
- Register: During the registering process a trigger event that communicates with the backend and stores user data in a database.
- Logout: Each session will last only for 1 hour, if exceeds one hour trigger event will take place and automatically display the login out.

Interoperability
Do you use the API https://marcconrad.com/uob/banana/api.php?

To ensure interoperability, I've integrated the provided Banana API to dynamically load a game image into the application I developed when the user fails to win using Fetch API in React.

API Routes are used for

- authentication (api/auth): Handle user login using JWT authentication, validates emails and passwords, and returns a JWT token upon successful login,
- Game History(api/history): Fetch the user's game history and allow the user to add new game records including game level and number of moves.
- User Registration(api/user): Register a new user with input validation for name, email, and password, check for existing users, hash a password and store it in the database, and register the user

Protocols Used:

- HTTP: for data exchange between the frontend and backend.
- JWT: secure the user authentication and API Access.

Virtual Identity
How did you implement virtual identity in your code? (e.g., Passwords, Cookies, IP Numbers …)

- Authentication with JWT: Virtual identity is managed using JWT tokens, which are generated upon successful user login or register. Tokens are stored in local storage.
- Bcrypt is used to hash passwords and ensure passwords are not stored as plain text in the database.
- Unique user identification: Each user is assigned a unique identifier in the database

Any other interesting features:

- Motivational quotes appear on the screen to motivate the user every 3 seconds.