CA2 Individual Report

|  |  |
| --- | --- |
| Name | Chai Jun Xuan |
| Student Id | P2336077 |
| Class | DIT/FT/1B/04 |
| Github Repository URL | <https://github.com/ST0503-BED/bed-ca2-junxuan000> |
| Github Account ID | junxuan000 |

For each competencies, find links to pull requests/commits/files that demonstrate the completion of the requirement. Replace each “**?**” with your Self Rating.

For Self Rating, you may rate yourself accordingly if you feel that you:

1. Have little or **no** understanding. and did not attempt the requirement
2. Have **limited** understanding of the specific competency
3. Have **basic** understanding and only able to replicate examples from tutorials/practicals.
4. Have **adequate** understanding and can extend from what you have learned to fulfil specifications.
5. Have **solid** understanding in the specific competency, able work on the requirement without much references.
6. Have **excellent** understanding and implemented the requirement according to latest industry guidelines, best practices and documentations.

**Important**

1. You are require to provide for each competency:
   * A brief **description**
   * **One or two** of your best implementations with URL **link** to respective repository request/commits/files.  
     **The implementations may come from Section A or B.**
   * You may also provide **screenshots** using POSTMAN to test API test.
2. You are to ensure the hyperlink in this document works. **Failure to do so will result in a 50% deduction of marks.**

|  |  |  |  |
| --- | --- | --- | --- |
| No. | Competencies | Describe What Was Done | Self Rating |
| 1 | Backend Server | *(How did you handles incoming requests, communicates with the database, and provides appropriate responses?)*  In the backend server, when a user makes a request, the system directs that request to specific areas based on the type of request and the information needed. Before and after handling the request, there are checks and processes in place, like making sure passwords are secure and managing user authentication tokens.  Controllers are like the traffic directors, guiding the flow of information. They handle specific types of requests, interacting with the database to get or update data. The database, where all the information is stored, is accessed through a service that manages these interactions.  Once the backend has the necessary information from the database, it puts together a response. This response is usually in a format like JSON (a way to structure data) and is sent back to the user. This whole process ensures that when a user asks for something, the backend finds the information, checks it, and sends back the right response.  <https://github.com/ST0503-BED/bed-ca2-junxuan000/commit/a9cfffece7b232d77a73cbffc3bd512e6cc30ee8>  This commit show when user click the button it will send a request to the server, the sever will then processing with the database in order to show the user list and the task list. | 5/5 |
| 2 | Functionality | *(Did your features implemented meets the specified requirements and fulfils its intended purpose?)*       1. **View Messages:**    * **Implementation Overview:**      + The code utilizes the **DOMContentLoaded** event to ensure that the script runs after the HTML content is fully loaded.      + It fetches messages from the server using the **fetchMethod** function and populates the **messageList** on the page.      + Each message is displayed in a card format, including message text, user ID, and creation timestamp.    * **Assessment:**      + The implementation appears to fulfill the intended purpose of retrieving and displaying messages.      + The structure of the card and the displayed information align with the standard expectations for presenting messages.      + The use of **responseStatus** and **responseData** in the callback suggests error handling and logging for better debugging. 2. **User Login:**    * **Implementation Overview:**      + The login functionality is triggered by the **DOMContentLoaded** event.      + It listens for form submissions, extracts the entered username and password, and sends a login request to the server.      + Upon successful login, the user is redirected to the "profile.html" page, and the token is stored in the local storage.      + In case of unsuccessful login, a warning card is displayed with an appropriate message.    * **Assessment:**      + The login implementation seems to meet the specified requirements.      + Successful logins redirect users to the profile page, and tokens are stored locally for subsequent authentication.      + Unsuccessful logins display a warning message, enhancing user feedback. 3. **Backend Integration:**    * The server-side handling involves a POST route in the **userController** that includes login, password comparison, token generation, and token sending.    * **Assessment:**      + The server-side integration appears comprehensive, including password hashing and token management.      + The chaining of middleware functions (**bcryptMiddleware.comparePassword** and **jwtMiddleware.generateToken**) suggests a well-structured authentication flow.   [*https://github.com/ST0503-BED/bed-ca2-junxuan000/commit/4d3ad923003a17ed5f891f9a1763b398de8e60bd*](https://github.com/ST0503-BED/bed-ca2-junxuan000/commit/4d3ad923003a17ed5f891f9a1763b398de8e60bd) | 5/5 |
| 3 | Code Quality | *(How did you organise your code to ensure maintainability, readability and adherence to coding best practices?)*  *I organized my code using a modular approach, breaking down tasks into event listeners, callbacks, and functions. This structure improves clarity and maintainability, helping me understand each part of the application easily. I followed consistent and clear naming conventions for variables and functions, contributing to overall code readability. The use of an event-driven architecture, along with well-implemented event listeners and callbacks, ensures a smooth execution flow and effective handling of user interactions. Conditional rendering based on user roles, especially the isAdmin check, creates a dynamic and responsive user interface tailored to different privilege levels. I used dynamic HTML creation and the Fetch API for asynchronous requests, making the code flexible and readable. While error handling is robust, ongoing improvements could further enhance the code's resilience. Comments provide valuable context and explanations, aiding developers in understanding specific code segments. In summary, my code demonstrates a commitment to best practices, fostering a maintainable and understandable codebase.*  [*https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/allTask.html*](https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/allTask.html)[*https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/allTask.js*](https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/allTask.js) | 5/5 |
| 4 | Modularity | *(How did you've organized your project to promote code reusability and maintainability?)*  *To enhance code reusability and maintainability, I adopted a modular structure in my project, particularly within the frontend and backend components. In the frontend JavaScript, I organized the code using the Document Object Model (DOM)ContentLoaded event to ensure proper initialization. I created separate event listeners and callbacks for distinct functionalities, such as fetching and displaying messages. The modularity is evident in the separation of concerns, allowing for easier maintenance and updates to specific features.*  *Additionally, I incorporated reusable functions like fetchMethod to handle asynchronous requests consistently. For message-related functionalities, I structured the code to dynamically create HTML elements based on fetched data, promoting flexibility and ease of understanding. The conditional rendering of buttons, such as the delete and update buttons, based on user privileges, demonstrates a modular approach catering to different scenarios.*  *In the backend, I modularized the code by separating the routes, controllers, and models for handling messages. Each module has dedicated functions for creating, reading, updating, and deleting messages, following the principles of encapsulation. This modular organization facilitates a clear separation of concerns, making it easier to extend or modify specific functionalities without affecting the entire codebase.*  *Moreover, the consistent use of callbacks ensures a smooth flow of asynchronous operations, promoting maintainability. By adopting these modular practices, my project achieves a balance between code reusability and maintainability, fostering a more comprehensible and scalable codebase.*    [*https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/chatAction.js*](https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/chatAction.js)[*https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/createMessage.js*](https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/createMessage.js)  [*https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/viewMessage.js*](https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/viewMessage.js)  [*https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/src/controllers/messageController.js*](https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/src/controllers/messageController.js) | 4/5 |
| 5 | Error Handling | *(How did you manage errors, provide informative feedback, and handle exceptional situations?)*  The code demonstrates a robust error-handling approach at multiple levels. On the frontend, forms for updating pet details employ event listeners to intercept submissions. After gathering input values, a fetch request is made, and any non-200 status or error responses trigger console logs for clear feedback. The callback function interprets server responses, distinguishing between client errors and successful updates, with dedicated functions displaying user-friendly messages. In the backend, the createQuestTrackerController ensures thorough checks before creating a QuestTracker, handling errors meticulously and logging meaningful messages. Both frontend and backend consistently provide informative error responses, such as a 400 status for insufficient pet points. The inclusion of well-placed console logs aids developers in debugging and comprehending the application's execution flow.  <https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/src/controllers/questTrController.js>  <https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/assignQuest.js>  <https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/updateEquipment.js> | 5/5 |
| 6 | Documentation | *(What was written for comments, readme and external documentation?)*  The README documentation provides a thorough and well-structured guide to understanding and setting up the CA2 project. It introduces the project's key sections, emphasizing features like authentication, user interaction, and player/pet/admin management. The folder structure is clearly outlined, highlighting key directories for better organization. Prerequisites, including Node.js and npm, are explicitly detailed. The cloning process is made accessible with step-by-step instructions, and setting up environment variables is meticulously explained, emphasizing security practices. The guide includes a comprehensive section on installing dependencies, initializing the database, and syncing changes with the remote repository. Additionally, the code comments within server and public files are detailed and informative, offering insights into conditions, errors, and the logic behind crucial functions. Overall, the documentation effectively balances clarity and depth, providing a comprehensive resource for developers engaging with the project.  <https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/README.md>  <https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/src/controllers/playerController.js>  <https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/editTask.js> | 5/5 |
| 7 | JWT for Authentication | *(How did you generate, validate, and manage user tokens?)*  The authentication mechanism in this codebase relies on JSON Web Tokens (JWT) for secure user login and access control. When a user logs in, their credentials are validated, and if successful, the user ID, email, and hashed password are stored for subsequent use. The generation of JWT involves creating a payload with essential user information, such as user ID, email, and timestamp, and signing it using the jsonwebtoken library. The resulting token, representing the user session, is then sent to the client. Middleware functions ensure the smooth flow of this process, including token generation, sending, and verification. JWT configuration details, like the secret key and expiration duration, are securely managed. Overall, this approach enhances user authentication, ensuring a secure and efficient way to manage user sessions and control access throughout the application.  <https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/src/middleware/jwtMiddleware.js> | 5/5 |
| 8 | BCrypt for Password Hashing | *(How did you securely store and validate user passwords)*  This project incorporates the bcrypt library to enhance the security of user passwords. When a user logs in, a middleware function called comparePassword checks if the provided password matches the securely hashed password stored in the system. If there's a match, the user is authenticated; otherwise, an authentication error is returned. Another middleware function, hashPassword, is employed during user registration to securely hash passwords before storage. This function uses bcrypt to create a strong hash with a defined number of salt rounds. The hashed password is then saved for future use. These mechanisms ensure that passwords are stored and validated securely, contributing to the overall robustness of the authentication system.  <https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/src/middleware/bcryptMiddleware.js> | 5/5 |
| 9 | Frontend to Backend HTTP Requests | *(How did you setup frontend communication with the backend?)*  The frontend communication with the backend is orchestrated through JavaScript event listeners and fetch requests. The DOMContentLoaded event triggers the setup of various functionalities. For example, in the "Add Task" section, the form submission event is intercepted to gather task details, which are then sent to the backend using the fetchMethod function with a POST request to the "/api/tasks" endpoint. The response status and data are logged for debugging, and the form fields are reset upon a successful response. Similar event listeners and fetch requests are employed in sections such as user data retrieval and quest assignment. These interactions provide a seamless integration between the frontend and backend, allowing for dynamic updates of user interfaces based on the server's responses. Additionally, error handling and informative messages are incorporated to enhance the user experience by conveying relevant feedback.  <https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/assignQuestForPet.html>  <https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/allUser.js>  <https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/addTask.js> | 5/5 |
| 10 | HTML & CSS Proficiency | *(Provide example of how you managed to create structured layouts and visually appealing styles)*  *The example codes showcases a well-organized and visually appealing webpage. It includes a responsive navigation bar, a nicely designed form for creating pets using Bootstrap, and a consistent color scheme managed in separate CSS files. The layout is structured with cards for content presentation, and the design is responsive, adjusting to different screen sizes. The use of external librarys like Bootstrap contributing to code modularity. Overall, the code demonstrates attention to detail, follows best practices, and effectively utilizes web development technologies to create a user-friendly and responsive interface.*  [*https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/css/color.css*](https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/css/color.css)  [*https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/css/style.css*](https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/css/style.css)[*https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/pvp.html*](https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/pvp.html) | 5/5 |
| 11 | Frontend Framework | *(What did you do to enhances the user experience and simplifies complex tasks?)*  To ensure a polished user experience, I incorporated Bootstrap 5 for styling and layout, providing a responsive and visually appealing design. The navigation bar includes convenient links for easy navigation, along with options for essential user actions like profile management, logout, and login. I took care to maintain a consistent color scheme throughout the application, contributing to a professional and visually pleasing aesthetic. Additionally, I added cursor feedback when hovering over buttons, enhancing user interaction. The "Add Task" section features a well-organized form with input fields for task title, description, and points. Leveraging JavaScript and Axios, I handled form submissions asynchronously. Robust error handling was implemented to manage scenarios where input fields are left empty. The code adheres to a modular approach, with concerns separated into different JavaScript files, promoting maintainability and readability.      [*https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/addTask.html*](https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/addTask.html)  [*https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/allTask.js*](https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/allTask.js) | 5/5 |
| 12 | JavaScript and DOM Manipulation | *(How does your frontend utilise dynamic content and DOM manipulation?)*  *The JavaScript code effectively employs dynamic content and DOM manipulation for a frontend web application. In one section, it fetches a list of pets from the server and dynamically generates a visually organized display using Bootstrap grid layout. Each pet's information, like name, species, points, and creation date, is dynamically inserted into HTML elements and added to the DOM. In another section, it fetches detailed information about a specific pet identified by the player\_id from the URL parameters. The response is used to create a card displaying comprehensive details, such as pet name, image, species, points, weapon, head, and armor. Interactive buttons for assigning quests, managing equipment, and engaging in pet vs. pet battles are included, each with an event listener for seamless navigation to related pages when clicked. This approach enhances user interaction and provides a visually appealing and responsive user interface.*  [*https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/petAction.js*](https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/petAction.js)  [*https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/allPet.js*](https://github.com/ST0503-BED/bed-ca2-junxuan000/blob/main/public/js/allPet.js) | 5/5 |