Internship Experience

10 Months

more of course + training

3 months HTML,CSS,JS + data structures/algo

3 Months started working on Angular

3 Months different documentations

1 months started working on drf+python  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
  
Introduction ->   
I have developed an e-commerce website using the MERN stack, where I focused primarily on backend development. The project aimed to create a seamless and secure online shopping experience for users.

**Technical Details:**

* **MERN Stack**: I utilized MongoDB, Express.js, React.js, and Node.js to build the application. This allowed for a robust and scalable architecture.
* **JWT Authentication**: To ensure secure access and protect user data, I implemented JSON Web Token (JWT) authentication. This helps in maintaining secure sessions and managing user roles and permissions effectively.
* **Braintree Payment System**: For handling payments, I integrated the Braintree payment gateway. This enabled secure transactions, supporting various payment methods like credit/debit cards and PayPal.

**Role and Contributions:**

* **Backend Development**: I designed and developed the backend API using Node.js and Express.js, ensuring efficient data management and seamless communication with the frontend.
* **Database Management**: I utilized MongoDB for efficient data storage and retrieval, creating schemas and models that ensured data integrity and performance.
* **Authentication and Authorization**: Implemented JWT for user authentication and authorization, securing endpoints and managing user sessions.
* **Payment Integration**: Integrated the Braintree payment system to handle transactions securely, providing a reliable payment experience for users.

**Outcome:**

This project enhanced my skills in backend development, authentication mechanisms, and payment gateway integration. It also improved my ability to work with the MERN stack and provided me with valuable experience in developing secure and scalable web applications.

1. **JWT Authentication ->   
   Basic Authentication**:
   * **Description**: Involves sending a username and password with each request. Credentials are usually encoded in base64.
   * **Pros**: Simple to implement.
   * **Cons**: Not secure unless used with HTTPS, as credentials are exposed with every request.
2. **Session-based Authentication**:
   * **Description**: After a user logs in, the server creates a session and stores it in memory or a database. The session ID is stored in a cookie and sent with each request.
   * **Pros**: Widely used and supported, easy to implement for small applications.
   * **Cons**: Not scalable due to server-side session storage, vulnerable to CSRF attacks.
3. **Token-based Authentication**:
   * **Description**: After logging in, the server generates a token (e.g., JWT) and sends it to the client. The client includes this token in the Authorization header of each request.
   * **Pros**: Stateless and scalable, no need to store sessions on the server, suitable for APIs.
   * **Cons**: Requires secure storage of tokens on the client side, handling token expiration and refresh can be complex.
4. **OAuth**:
   * **Description**: An open standard for access delegation, commonly used for token-based authorization. Allows users to grant third-party access to their resources without sharing credentials.
   * **Pros**: Secure, widely adopted, allows for granular permissions.
   * **Cons**: Complex to implement, requires understanding of different OAuth flows.

**Why I Chose JWT**

* **Stateless**: JWT is stateless, meaning the server does not need to store session data. This makes it scalable and suitable for distributed systems.
* **Secure**: JWT can be securely signed and optionally encrypted, ensuring data integrity and confidentiality.
* **Versatile**: JWT can be easily used across different domains and platforms, making it ideal for microservices and mobile applications.
* **Performance**: JWTs are self-contained and reduce the need for frequent database lookups to authenticate users.

**How JWT work ?**In simple authentication we can gesalt and then hash the password

 **1xx Informational**:

* **100 Continue**: The server acknowledges that the initial part of the request has been received and the client can continue with the request.
* **101 Switching Protocols**: The server is switching protocols as requested by the client (e.g., upgrading to WebSocket).

 **2xx Success**:

* **200 OK**: The request was successful, and the server has returned the requested resource.
* **201 Created**: The request has been fulfilled, and a new resource has been created as a result.
* **202 Accepted**: The request has been accepted for processing, but the processing has not been completed.
* **204 No Content**: The server successfully processed the request but is not returning any content (often used for DELETE operations).

 **3xx Redirection**:

* **301 Moved Permanently**: The requested resource has been permanently moved to a new location. The client should use the new URL provided in the response.
* **302 Found / 307 Temporary Redirect**: The requested resource has been temporarily moved to a different location. The client should use the new URL provided in the response (302 historically, 307 preserves the HTTP method).

 **4xx Client Error**:

* **400 Bad Request**: The server could not understand the request due to malformed syntax, missing parameters, or invalid data.
* **401 Unauthorized**: The request requires authentication. The client needs to provide valid credentials for access.
* **403 Forbidden**: The server understood the request but refuses to authorize it. The client does not have permission to access the resource.
* **404 Not Found**: The server cannot find the requested resource. This is a common response for requests to non-existent URLs.

 **5xx Server Error**:

* **500 Internal Server Error**: The server encountered an unexpected condition that prevented it from fulfilling the request.
* **502 Bad Gateway**: The server received an invalid response from an upstream server while acting as a gateway or proxy.
* **503 Service Unavailable**: The server is currently unable to handle the request due to temporary overloading or maintenance.

1. **Collect Payment Information**:
   * Your frontend collects payment details from the user, such as credit card information or PayPal account details.
2. **Generate Payment Token**:
   * Instead of directly sending sensitive payment information to your backend, you securely send this information to Braintree's servers using their client-side SDK (Software Development Kit).
   * Braintree processes this information and returns a unique payment token to your frontend. This token represents the payment method (e.g., credit card) and is tied to the customer's session.
3. **Send Payment Token to Backend**:
   * Once you receive the payment token from Braintree, securely send it to your backend server along with any other necessary transaction details (e.g., amount, currency).

**Backend Flow:**

1. **Receive Payment Token**:
   * Your backend server receives the payment token and transaction details from the frontend.
2. **Initiate Payment Transaction**:
   * Using Braintree's server-side SDK or API, your backend server uses the received payment token to initiate a payment transaction request to Braintree.
   * This transaction request includes the payment token, amount to charge, and any other relevant details.
3. **Handle Transaction Response**:
   * Braintree processes the transaction request and returns a response to your backend server indicating whether the payment was successful or not.
   * The response typically includes a transaction ID that uniquely identifies the payment transaction.
4. **Verify Transaction Status**:
   * After receiving the response from Braintree, your backend server verifies the status of the transaction (e.g., whether it was successful, pending, failed).
5. **Update Payment Status**:
   * Based on the transaction status received from Braintree, update your backend database or application to reflect the payment status for the specific transaction identified by the payment token.
   * Mark the transaction as paid or handle any errors or retries as necessary.

As a JavaScript developer for this project, my role involved:

1. \*\*Managing Game Assets:\*\* I handled loading and managing various game assets such as images and audio files required for the game's visuals and sound effects.

2. \*\*Game Mechanics Implementation:\*\* I implemented core game mechanics including paddle movement, ball physics, collision detection with walls, paddle, and bricks, as well as functionalities like pausing the game.

3. \*\*Dynamic Gameplay Elements:\*\* I designed dynamic elements such as bricks with varied characteristics (e.g., changing paddle size, altering ball speed) to enhance gameplay diversity and challenge.

4. \*\*User Interface (UI) Updates:\*\* I updated and displayed game statistics such as score, lives remaining, and current level using HTML5 canvas, ensuring a responsive and engaging user interface.

5. \*\*Game Flow Control:\*\* I managed game flow logic including handling game over conditions, level transitions, and win scenarios, providing a seamless and enjoyable player experience.

6. \*\*Audio Management:\*\* I integrated and managed sound effects throughout the game, allowing players to toggle sound on/off as per their preference.

7. \*\*Bug Fixing and Optimization:\*\* I ensured smooth performance by identifying and fixing bugs, optimizing code where necessary to maintain consistent frame rates and responsiveness.

Overall, my role as a JavaScript developer was pivotal in creating a functional and entertaining game experience, combining technical skills with creative problem-solving to deliver a polished end product.