**COMMENTS DURING DEVELOPMENT**

**Highlights:**

1. **Modular Structure:** The project follows a modular structure with separate modules for characters, DTOs, and services. This design promotes maintainability and scalability.
2. **Validation with class-validator:** The use of class-validator for DTO validation ensures that incoming data is validated against predefined rules, enhancing the reliability of the application.
3. **API Documentation:** Leveraging Swagger, the API is automatically documented, providing developers with an interactive and easy-to-use documentation interface.
4. **Error Handling:** Exception handling simplifies the process of managing errors, making the code more readable and promoting a consistent approach to error responses.
5. **Database Interaction with TypeORM:** TypeORM is used to interact with the PostgreSQL database, allowing for a seamless integration between the application and the database.
6. **User Experience:** Emphasis was placed on creating a user-friendly experience with an auto-suggest input field, character selection, and clear feedback on selections.
7. **API Integration: L**everaged the Marvel API for real-time character data, showcasing integration skills.

**Challenges:**

1. **Styling Without Frameworks:** Omitting UI frameworks and CSS libraries meant manually styling components for consistent and visually appealing designs.
2. **Testing NestJS Controllers:** Testing controllers, especially those involving database interactions, can be challenging. Mocking dependencies and setting up test databases are essential for comprehensive test coverage.
3. **DTO and Validation Overhead:** While DTOs and validation are crucial for maintaining a structured API, they introduce some overhead in terms of code verbosity. Striking a balance between validation and code conciseness is important.
4. **Database Schema Changes:** Adapting the database schema to meet evolving requirements may require careful consideration to avoid data migration issues.

**Gotchas:**

1. **NestJS Dependency Injection:** Understanding the intricacies of NestJS dependency injection is important, especially when dealing with services and controllers. It's crucial to grasp the lifecycle and scope of injected components.
2. **DTO Naming Conventions:** Ensuring consistent naming conventions for DTOs and maintaining a clear relationship between DTOs and entities can prevent confusion in the long run.
3. **API Key Handling:** Securely managing and hiding API keys to prevent misuse and unauthorized access

**Decisions Made:**

1. **Swagger for API Documentation:** Swagger is selected for its ease of integration with NestJS, providing a self-updating API documentation interface.
2. **Class-Validator for DTO Validation:** The decision to use class-validator for DTO validation is made to enforce strict data validation and maintain clean, readable DTO classes.
3. **Separation of Concerns:** The modular structure emphasizes the separation of concerns, allowing for easier maintenance and testing.