**Overview**

The Rick & Morty API serves as an intermediary for accessing character data from the Rick and Morty API, providing methods to retrieve characters based on various filters or a specific ID. This API employs a layered architecture including controllers, services, repositories, models (DTOs), and middleware, ensuring a clean separation of concerns and scalability without compromising simplicity.

**Core Design Principles and Practices**

* **SOLID Principles**: Each component of the API adheres to the SOLID principles, ensuring that the system is easy to maintain and extend.
* **DRY (Don't Repeat Yourself)** and **KISS (Keep It Simple, Stupid)**: The codebase avoids redundancy and maintains simplicity, focusing on delivering functionality without unnecessary complexity.
* **YAGNI (You Aren't Gonna Need It)**: Features or structures that are not currently needed are not included, keeping the API lightweight and efficient.
* **Domain-Driven Design (DDD)**: The architecture reflects a clear and modular domain model, aligning closely with the business domain of interacting with the Rick and Morty API.
* **Dependency Injection**: Used throughout the application to manage dependencies, facilitating loose coupling and easier management of cross-cutting concerns.

**Implementation Choices and Simplifications**

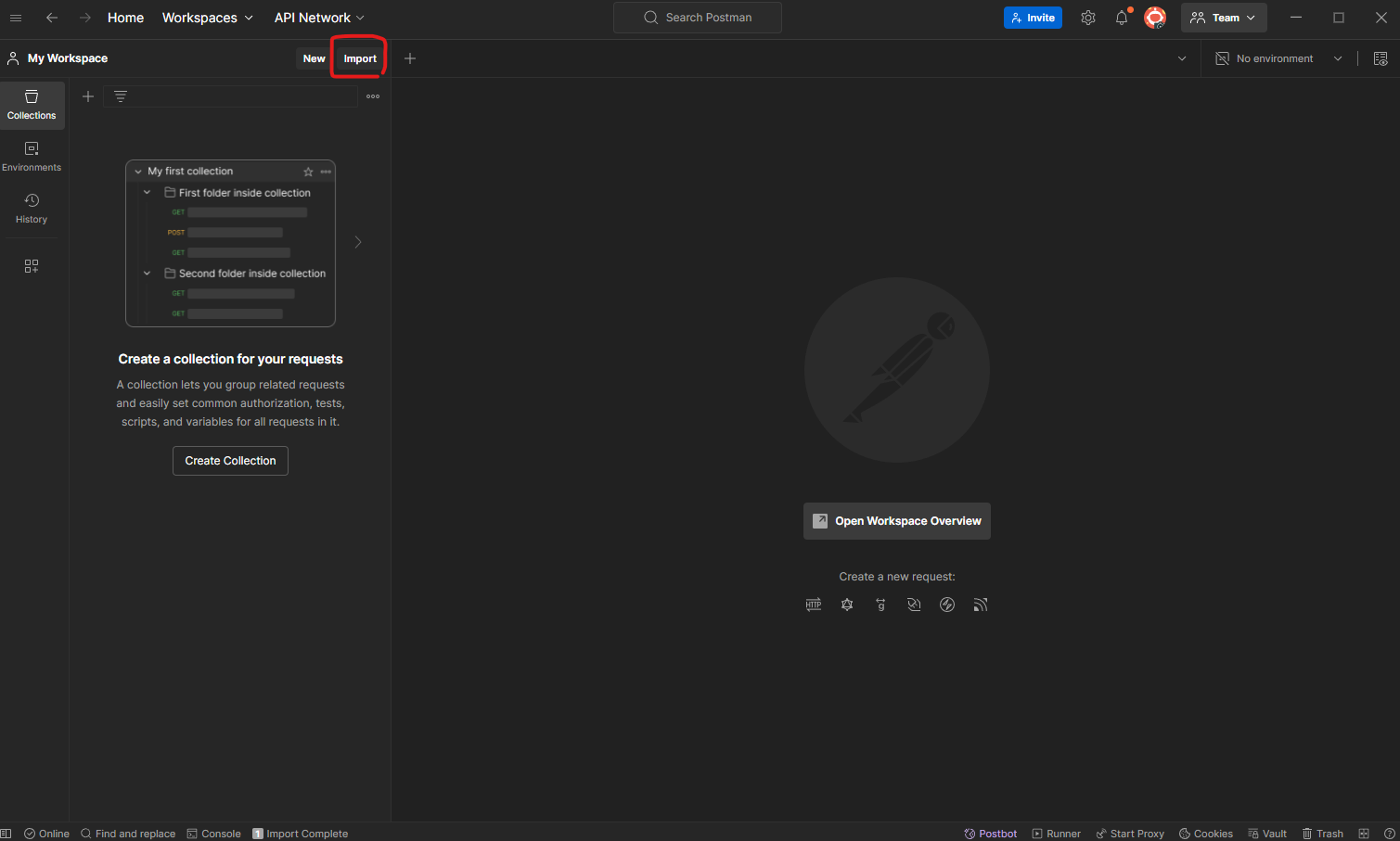
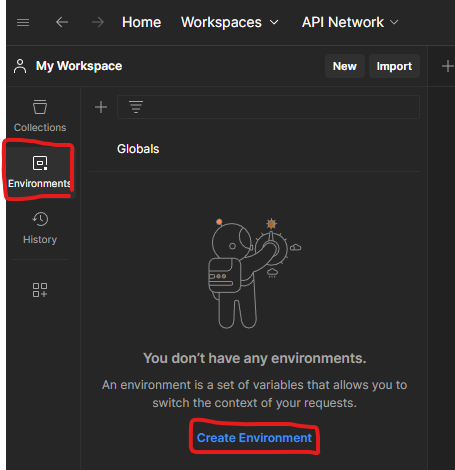
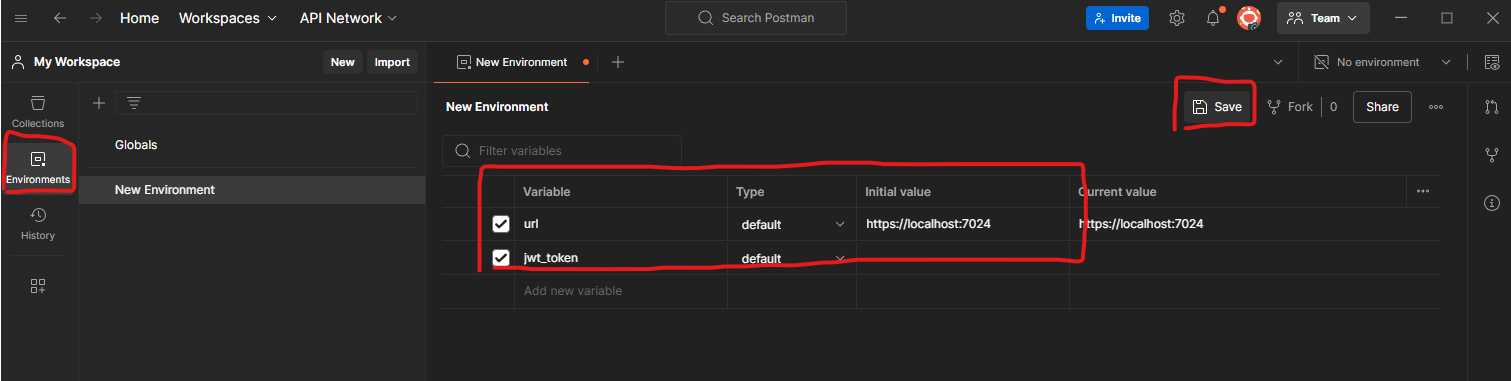
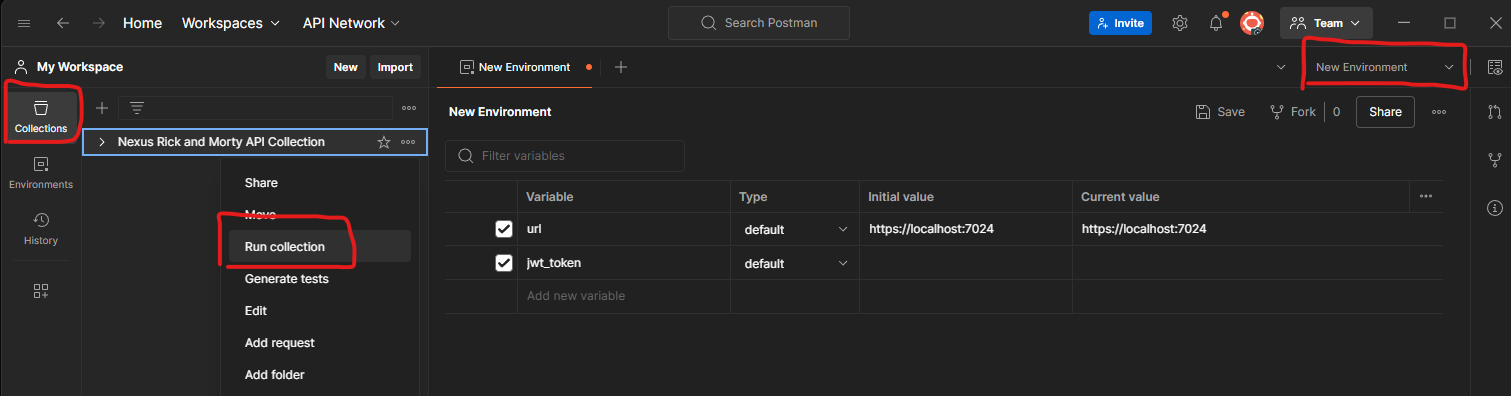
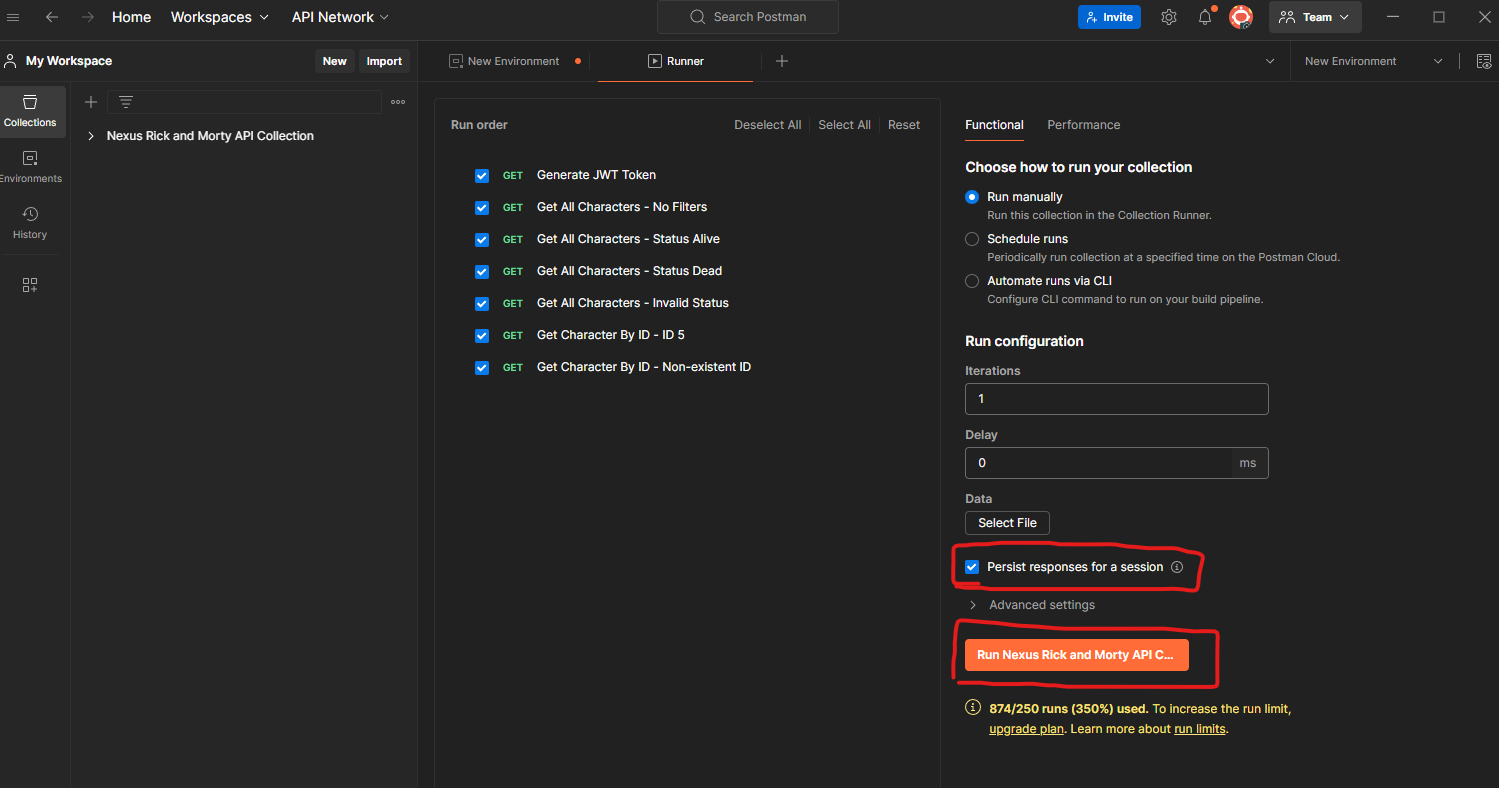
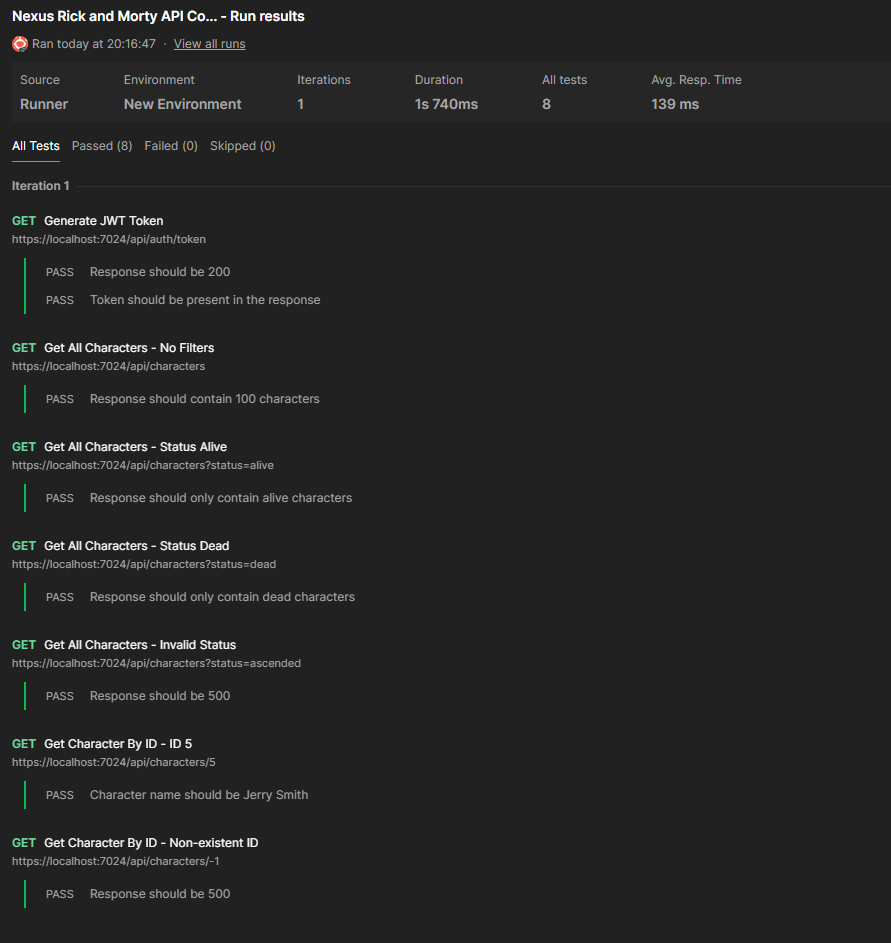
In keeping with the principles of YAGNI and KISS, the API:

* **Avoids Over-engineering**: The codebase is straightforward, without layered DTOs, extensive mappings, factories, or other scalability features that might complicate the codebase unnecessarily.
* **Uses Direct Data Transfer Objects (DTOs)**: The DTOs used are direct mappings to API responses without intermediate transformations, ensuring clarity and reducing overhead.
* **Minimizes Abstractions**: Only essential abstractions are present to meet the current requirements, avoiding speculative generality.

**Testing Choices**

* **Regression Tests – Postman**: The API functionality can be tested using the provided Postman collection named Rick-And-Morty-Postman-Collection.json.
* **Unit Tests**: There are no Unit Tests given the application's simplicity, Unit Testing would require more time to develop than the application itself, as it would require Dependency Mocking, HTTP Factories and more.

**How to: Postman Testing**

* To run the Postman Collection, follow these screenshots.
  + Import the collection 
  + Create the Environment 
  + Create the variables “url” and “jwt\_token”. The “url” should be set to the API URL and Port. The “jwt\_token” should be empty. Then click Save. 
  + Go back to Collections, change the Environment to the new one, right click the collection and press Run Collection. 
  + Click on Persist Responses if you want to view the logs and click Run. The Colleciton will request and store the Token from the API automatically. 
  + This is how the Tests should look if the API is working. 

**Architecture and Components**

1. **Controllers (Presentation)**:
   * Handle HTTP requests and delegate business operations to the service layer.
   * Ensure security with JWT-based authentication, encapsulating user identity and claims handling.
2. **Business Logic**:
   * Orchestrates operations between the web interface and data repository.
   * Uses the repository pattern to decouple data access logic from business logic.
3. **Data Access (Repository)**:
   * The repository layer interacts directly with the Rick and Morty API, abstracting the specifics of HTTP communication from the service layer.
4. **Middleware**:
   * Custom middleware for exception handling standardizes API error responses.
   * Logging middleware provides insights into request handling and performance. The logs are stored into 10 MB txt files on the executable directory.
5. **Dependency Injection**:
   * Extensively used to wire up services, controllers, and other components at runtime, configured in the Startup class.
6. **Configuration and Setup**:
   * Configures services and middleware in the Program.cs, including routing, authentication, and API documentation tools (Swagger) (https://localhost:7024/swagger/index.html).

**Data Transfer Objects (DTOs)**

**ExternalApiResponse**

* **Info** (Info): Meta-information about pagination.
* **Results** (List<Character>): A list of character objects.

**Character**

* **Id** (int): Character identifier.
* **Name** (string): Name of the character.
* **Status** (string): Character's life status.
* **Species** (string): Species of the character.
* **Type** (string): Type or classification of the character.
* **Gender** (string): Gender of the character.
* **Origin** (Origin): Origin location of the character.
* **Location** (Location): Current location of the character.
* **Image** (string): URL of the character's image.
* **Episode** (List<string>): List of episodes featuring the character.
* **Url** (string): API endpoint URL for this character.
* **Created** (string): Timestamp of character creation.

**Info**

* **Count** (int): Total count of characters.
* **Pages** (int): Number of available pagination pages.
* **Next** (string): URL of the next page.
* **Prev** (string): URL of the previous page.

**Location and Origin**

* **Name** (string): Name of the location.
* **Url** (string): API URL for the location.

**ErrorResponse**

* **StatusCode** (int): HTTP status code of the error.
* **Message** (string): Error message.
* **Timestamp** (DateTime): Timestamp of the error.

**Controllers**

**CharactersController**

* **GetAllCharacters**:
  + Retrieves characters based on filters such as name, status, species, type, and gender.
  + Returns 200 OK with a list of characters or 500 Internal Server Error.
* **GetCharacter**:
  + Retrieves a character by their ID.
  + Returns 200 OK with a character or 500 Internal Server Error.

**AuthController**

* **GenerateToken**:
  + Generates a JWT token for user authentication.
  + Returns 200 OK with a JWT token string or 500 Internal Server Error.

**Middleware**

**ExceptionHandlingMiddleware**

* Catches and logs exceptions.
* Standardizes error responses across the API.

**LoggingHandlingMiddleware**

* Logs incoming requests and the time taken to process them.

**Business Logic**

**CharacterService (Implements ICharacterService)**

* **GetAllCharactersAsync**:
  + Retrieves characters matching specified filters until 100 are gathered or no more are available.
* **GetCharacterByIdAsync**:
  + Retrieves a single character by ID.

**Repository**

**CharacterRepository (Implements ICharacterRepository)**

* **HttpQuery**:
  + Executes an HTTP GET request to a specified URL and returns the response content.

**Startup and Configuration**

* Configures services, middlewares, and JWT authentication.
* Sets up Swagger for API documentation and testing in a development environment.

**Dependencies and External Configurations**

* Uses log4net for logging across components.
* HttpClient is configured to interact with the Rick and Morty API.