Assumptions:

- Fields: The required fields for user registration are assumed to be firstName, lastName, email, and password, along with an additional confirmPass field for password confirmation.
- Document Structure: The user document stored in the database includes fullName (a concatenation of firstName and lastName), the hashed password, and the email.
- JWT/Authorization: Access to endpoints requires an authorization token, which is generated using JWT during user login.

Approach:

I have employed a Layered Architecture, which organizes the application into distinct layers for better separation of concerns:

- Routes: This layer defines the application endpoints.
- Controller: This layer handles the response logic, including status codes and response content.
- Service: This layer contains the business logic, including validation and error handling.
- Persistence: This layer interacts directly with the database (MongoDB in this case) to perform CRUD operations and retrieve data.
- Model: This layer defines the schema and structure of the documents to be stored in the database.

Application Entry Point:

- app.js: This is the main file of the API. It serves as the core component that connects all other layers and configurations, initializing the application, setting up middleware, and establishing routes.

Authorization:

- I decided to use JWT as middleware with the verifyToken function. When a user logs in, the server creates an auth-token, which is saved in the headers. When a user tries to access an endpoint (e.g., /getUser), the server verifies this token before responding. If the token in the headers is valid, the user gains access; otherwise, they receive an "Access denied" message.