**Design Choices**

**RESTful Architecture**: The API is designed following RESTful principles, ensuring a standard and predictable interface for clients. This choice allows for scalability, flexibility, and ease of understanding.

**Express.js Framework**: Express.js is chosen as the server framework due to its simplicity, robustness, and widespread adoption within the Node.js ecosystem. It provides middleware support, routing capabilities, and facilitates the development of RESTful APIs.

**JSON Data Format**: JSON is used as the data interchange format due to its lightweight nature, readability, and ease of parsing in JavaScript. It provides a simple and efficient way to represent data transferred between the client and server.

**Challenges**

**Data Validation**: Ensuring the integrity and validity of data inputs is a significant challenge. It's crucial to validate user inputs to prevent unexpected behavior and security vulnerabilities. This includes validating request bodies, query parameters, and URL paths.

**Error Handling**: Handling errors gracefully and providing informative error messages is essential for a robust API. Identifying and handling various error scenarios, such as invalid requests, database errors, and authentication failures, can be challenging but necessary for a user-friendly experience.

**Resolutions**

**Input Validation Middleware**: Implement middleware functions to validate request bodies, query parameters, and URL paths. Use libraries like Joi or express-validator to define validation schemas and sanitize inputs to prevent injection attacks.

**Error Middleware**: Develop error-handling middleware to catch and handle errors at the global level. Use try-catch blocks to capture synchronous errors and leverage async/await or Promise.catch() for asynchronous operations. Return appropriate HTTP status codes and error messages to clients.

**Authentication and Authorization Middleware**: Integrate authentication middleware (e.g., Passport.js) to verify user identity and enforce access control based on user roles and permissions. Utilize JSON Web Tokens (JWT) for stateless authentication and implement role-based access control (RBAC) to restrict access to sensitive endpoints.

Overall, by adhering to RESTful principles, leveraging appropriate frameworks and libraries, and implementing robust security measures, the Bookstore API can provide a secure, scalable, and user-friendly experience for clients while effectively managing books in a bookstore.