Section 1: API Design

✓ **API Concept**

The Book Library Management System API will allow users to manage book information through standard CRUD operations (Create, Read, Update, Delete). Users can add new books, retrieve details, update book information, and delete books.

✓ **Endpoint Design**

The following API endpoints are defined:

- `GET /books`: Retrieve a list of all books in the library.
- `GET /books/{id}`: Retrieve a specific book by its unique ID.
- `POST /books`: Add a new book to the library.
- `PUT /books/{id}`: Update the details of a specific book.
- `DELETE /books/{id}`: Delete a specific book by ID.

✓ **Request and Response Design**

The request format and response format for each of the endpoints are as follows:

- `GET /books`: Returns a JSON array containing information about all books.
- `GET /books/{id}`: Returns a JSON object containing information about a specific book.
- `POST /books`: Accepts a JSON object containing information about a new book to be added to the library. Returns a JSON object containing information about the newly added book.
- `PUT /books/{id}`: Accepts a JSON object containing updated information about a specific book. Returns a JSON object containing information about the updated book.
- `DELETE /books/{id}`: Deletes a specific book by ID. Returns a JSON object containing information about the deleted book.

✓ **API Documentation**

Request and Response Design:

➢ **GET /books**

Returns a JSON array containing information about all books in the library.

Request:

- Method: `GET`
- URL: `/books`

Response:

```
[
    {
        "id": 1,
        "title": "The Great Gatsby",
        "author": "F. Scott Fitzgerald",
        "year": 1925,
        "publisher": "Charles Scribner's Sons"
    },
    {
        "id": 2,
        "title": "To Kill a Mockingbird",
        "author": "Harper Lee",
        "year": 1960,
        "publisher": "J. B. Lippincott & Co."
    }
]
```

- **GET /books/{id}**

Returns a JSON object containing information about a specific book.

Request:

- Method: `GET`
- URL: `/books/{id}`

Response:

```
{
    "id": 1,
    "title": "The Great Gatsby",
    "author": "F. Scott Fitzgerald",
    "year": 1925,
    "publisher": "Charles Scribner's Sons"
}
```

- **POST /books**

Accepts a JSON object containing information about a new book to be added to the library. Returns a JSON object containing information about the newly added book.

Request:

- Method: `POST`
- URL: `/books`
- Request Body: `JSON`

```
{
    "title": "The Catcher in the Rye",
    "author": "J. D. Salinger",
    "year": 1951,
    "publisher": "Little, Brown and Company"
}
```

Response:

Status: 201 Created (with the newly created book object in the response body)

```
{
    "id": 3,
    "title": "The Catcher in the Rye",
    "author": "J. D. Salinger",
    "year": 1951,
    "publisher": "Little, Brown and Company"
}
```

- **PUT /books/{id}**

Accepts a JSON object containing updated information about a specific book. Returns a JSON object containing information about the updated book.

Request:

- Method: `PUT`
- URL: `/books/{id}`
- Request Body: `JSON (fields to be updated)`

```
{
    "title": "The Catcher in the Rye",
    "author": "J. D. Salinger",
    "year": 1951,
    "publisher": "Little, Brown and Company"
}
```

Response:

Status: 200 OK (with the updated book object in the response body)

```
{
    "id": 3,
    "title": "The Catcher in the Rye",
    "author": "J. D. Salinger",
    "year": 1951,
    "publisher": "Little, Brown and Company"
}
```

- **DELETE /books/{id}**

Deletes a specific book by ID. Returns a JSON object containing information about the deleted book.

Request:

- Method: `DELETE`
- URL: `/books/{id}`

Response:

```
{
    "id": 3,
    "title": "The Catcher in the Rye",
    "author": "J. D. Salinger",
    "year": 1951,
    "publisher": "Little, Brown and Company"
}
```

If a page is not found, a `404 Not Found` error will be returned.

Potential edge cases for each API endpoint:

1. GET /books: Retrieve a list of all books in the library.

- Edge Case 1: If there are no books in the library, the response should be an empty JSON array.
- Edge Case 2: Handle pagination if there is a large number of books. Include query parameters like `page` and `limit` for controlling the number of books returned per request.

2. GET /books/{id}: Retrieve a specific book by its unique ID.

- Edge Case 1: If the specified book ID does not exist, return a 404 Not Found error.
- Edge Case 2: If the ID provided is not a valid format (e.g., non-numeric), return a 400 Bad Request error.

3. POST /books: Add a new book to the library.

- Edge Case 1: If the request body is empty or does not contain the required fields (title, author, year, publisher), return a 400 Bad Request error.
- Edge Case 2: If a book with the same title and author already exists, return a 409 Conflict error to indicate a duplicate entry.

4. PUT /books/{id}: Update the details of a specific book.

- Edge Case 1: If the specified book ID does not exist, return a 404 Not Found error.
- Edge Case 2: If the request body is empty, return a 400 Bad Request error.
- Edge Case 3: If the request body contains fields not allowed to be updated, ignore those fields and proceed with the update.

5. DELETE /books/{id}: Delete a specific book by ID.

- Edge Case 1: If the specified book ID does not exist, return a 404 Not Found error.
- Edge Case 2: After successful deletion, if the client tries to retrieve the same book ID again, return a 404 Not Found error.

6. Error Handling:

- Edge Case 1: If a client attempts an invalid endpoint (e.g., `/books/invalidEndpoint`), return a 404 Not Found error.
- Edge Case 2: If an unsupported HTTP method is used (e.g., POST for a read-only operation), return a 405 Method Not Allowed error.
- Edge Case 3: If the server encounters an internal error, return a 500 Internal Server Error with an appropriate error message.