# System Requirements Specification Index

For

# Library Management System

Version 1.0

**IIHT Pvt. Ltd.**

fullstack@iiht.com

# Table of Contents

# Library Management System

## System Requirements Specification

## 1 Project Abstract

The City Public Library System requires a streamlined library management application to digitize their operations. The system will track books, library members, and handle borrowing operations. It will enable librarians to efficiently manage the library's collection by categorizing books, processing member registrations, and handling book checkouts and returns. This system provides an organized way for library staff to manage resources and serve community members effectively.

## 2 Business Requirements:

| | |
|---|---|
| Screen Name | Console input screen |
| Problem Statement | 1. System needs to store and manage different types of library data (books and members)<br>2. System must support operations such as book checkout, return, and member registration<br>3. Console should implement object oriented concepts like inheritance and method overriding to achieve desired outcome |

# 3 Constraints

## 3.1 Class Requirements

1. `Book` Class:
    - Attributes: book_id, title, author, genre, publication_year, is_available
    - Methods: display_info(), checkout(), return_to_library()
    - Example: `Book("B001", "To Kill a Mockingbird", "Harper Lee", "Fiction", 1960, True)`
2. `Member` Class:
    - Attributes: member_id, name, email, books_borrowed
    - Methods: display_info(), borrow_book(), return_book()
    - Example: `Member("M001", "John Smith", "john@example.com", [])`
3. `FictionBook` Class (inherits from `Book`):
    - Additional attributes: fiction_type
    - Override methods: display_info()
    - Example: `FictionBook("B001", "To Kill a Mockingbird", "Harper Lee", "Fiction", 1960, "Novel")`
4. ‘NonFictionBook` Class (inherits from `Book`):
    - Additional attributes: subject
    - Override methods: display_info()
    - Example: `NonFictionBook("B002", "A Brief History of Time", "Stephen Hawking", "Non-Fiction", 1988, "Physics")`
5. `Library` Class:
    - Attributes: name, address, books, members

- Methods: add_book(), add_member(), checkout_book(), return_book(), get_available_books(), search_book_by_title(), search_book_by_author()
- Static methods: get_book_count(), get_member_count()
- Example: `Library("City Public Library", "123 Main St, Anytown")`

### 3.2 Operation Constraints

1. Book Checkout:
    - Member must exist in the system
    - Book must be available
    - Member cannot exceed maximum borrowing limit (3 books)
    - Must update book availability status
2. Book Return:
    - Book and member must exist in the system
    - Must update book availability status
    - Member must have borrowed the book
3. Member Registration:
    - Member ID must be unique
    - Email must be valid format (must contain @ and a domain)
4. Book Addition:
    - Book ID must be unique
    - Publication year must be a valid year (not in the future)
    - Book must be assigned to the correct subclass (Fiction/NonFiction)
5. Exception Handling:
    - Must handle BookNotFoundException
    - Must handle MemberNotFoundException

- Must handle BookNotAvailableException
- Must handle MaxBooksExceededException
- Must handle InvalidInputException

6. Object-Oriented Requirements:
   - Must use proper encapsulation (private attributes with getters/setters)
   - Must implement inheritance for book types
   - Must use polymorphism with method overriding
   - Must implement static methods and class variables

### 3.3 Output Constraints

1. Display Format:
   - Book information: display ID, title, author, genre, publication year, availability status
   - Member information: display ID, name, email, books borrowed
   - Each item must be displayed on a new line with proper formatting

2. Output Format:
   - Must show in this order:
     - § Show "== LIBRARY MANAGEMENT SYSTEM =="
     - § Show "Library Name: {name}"
     - § Show "Address: {address}"
     - § Show "Total Books: {count}"
     - § Show "Total Members: {count}"
     - § Show "Current Book Collection:"
     - § Show books with format: "{id} | {title} by {author} | {genre} | {year} | Available: {status}"
     - § Show "Search Results:" when displaying search results

# 4. Template code Structure:

**1.** Book Classes:

- `Book` (base class)
- `FictionBook` (derived class)
- `NonFictionBook` (derived class)

**2.** Member Class:

- `Member`

**3.** Library Class:

- `Library`

**4.** Exception Functions:

- `BookNotFoundException`
- `MemberNotFoundException`
- `BookNotAvailableException`
- `MaxBooksExceededException`
- `InvalidInputException`

**5.** Program Control:

- `main()` - main program function

# 5. Detailed Function Structure:

## 5.1 Book Class Implementation

1. Write a Python class to represent a book in the library system.

Define: **class Book:**

This function should:

- Accept parameters: book_id, title, author, genre, publication_year, is_available=True
- Validate publication_year is integer: if not isinstance(publication_year, int): raise ValueError("Publication year must be an integer")
- Validate publication_year not in future: if publication_year > datetime.datetime.now().year: raise ValueError("Publication year cannot be in the future")
- Initialize private attributes using double underscore: self.__book_id = book_id
- Implement @property decorators for all attributes: @property def book_id(self): return self.__book_id
- Implement is_available setter: @is_available.setter def is_available(self, value): self.__is_available = value
- Implement checkout() method: Check if available, set to False, return True/False
- Implement return_to_library() method: Check if checked out, set to True, return True/False
- Implement display_info() method: Return formatted string with all book details
- Example format: "B001 | To Kill a Mockingbird by Harper Lee | Fiction | 1960 | Available"

2. Write a Python class for fiction books that inherits from Book.

Define: **class FictionBook(Book):**

This function should:

- Accept additional parameter: fiction_type
- Call parent constructor: `super().__init__(book_id, title, author, genre, publication_year, is_available)`
- Initialize private fiction_type attribute: `self.__fiction_type = fiction_type`
- Implement @property for fiction_type: `@property def fiction_type(self): return self.__fiction_type`
- Override display_info() method: Get base info with `super().display_info()` and append fiction type
- Example format: "B001 | To Kill a Mockingbird by Harper Lee | Fiction | 1960 | Available | Type: Novel"

3. Write a Python class for non-fiction books that inherits from Book.

Define: **class NonFictionBook(Book):**

This function should:

- Accept additional parameter: subject
- Call parent constructor: `super().__init__(book_id, title, author, genre, publication_year, is_available)`
- Initialize private subject attribute: `self.__subject = subject`
- Implement @property for subject: `@property def subject(self): return self.__subject`
- Override display_info() method: Get base info with `super().display_info()` and append subject
- Example format: "B002 | A Brief History of Time by Stephen Hawking | Non-Fiction | 1988 | Available | Subject: Physics"

## 5.2 Member Class Implementation

4. Write a Python class to represent a library member.

Define: **class Member:**

This function should:

- Accept parameters: member_id, name, email, books_borrowed=None
- Validate email format with multiple checks:
  - Must contain '@': `if not '@' in email:`
  - Cannot start with '@': `if email.startswith('@'):`
  - Must have domain with dot: `if '.' not in email.split('@')[1]:`
  - Check for empty domain parts and short TLDs as shown in solution
- Initialize private attributes: `self.__member_id = member_id`
- Handle default books_borrowed: `self.__books_borrowed = books_borrowed if books_borrowed is not None else []`
- Implement @property decorators for all attributes
- Implement books_borrowed property: Return copy to prevent modification: `return self.__books_borrowed.copy()`

- Implement borrow_book() method: Check limit (3), check book availability, call book.checkout(), add to list
- Implement return_book() method: Check if book in borrowed list, call book.return_to_library(), remove from list
- Implement display_info() method: Return formatted string with member details and book count

## 5.3 Library Class Implementation

**5.** Write a Python class to represent the library system.

Define: **class Library:**

This function should:

- Define class variables: book_count = 0 and member_count = 0
- Accept parameters: name, address
- Initialize private attributes and empty dictionaries: self.__books = {}, self.__members = {}
- Implement @property decorators for name and address
- Implement static methods for counts: @staticmethod def get_book_count(): return Library.book_count
- Implement add_book() method: Check for duplicates, add to dictionary, increment count
- Implement add_member() method: Check for duplicates, add to dictionary, increment count
- Implement checkout_book() method: Validate book and member exist, call member.borrow_book()
- Implement return_book() method: Validate book and member exist, call member.return_book()
- Implement get_available_books() method: Use dictionary comprehension to filter available books
- Implement search methods with None validation and case-insensitive matching
- Implement get_book() and get_member() methods: Return object or None if not found
- Implement get_all_books() and get_all_members() methods: Return copies of dictionaries

**6.** Write search methods for the Library class.

Define: **search_book_by_title(self, title) and search_book_by_author(self, author)**

This method should:

- Validate None input: `if title is None: raise ValueError("Search title cannot be None")`
- Convert search term to lowercase: `title = title.lower()`
- Use dictionary comprehension for case-insensitive search
- For title search: `{book_id: book for book_id, book in self.__books.items() if title in book.title.lower()}`
- For author search: Use word splitting for partial matches: `if author in book.author.lower().split()`
- Return dictionary of matching books

**7.** Write get methods for the Library class.

Define: **get_book(self, book_id) and get_member(self, member_id)**

This method should:

- Include debug prints as shown in skeleton: `print(f"get_book called with book_id: {book_id}")`
- Use explicit key checking: `result = None if book_id not in self.__books else self.__books[book_id]`
- Print result: `print(f"get_book returning: {result}")`
- Return the result (object or None)

### 5.4 Main Program Function

**7.** Write a Python function to demonstrate the complete library management system.

Define: **display_inventory_summary(summary)**

This function should:

- Accept summary dictionary parameter
- Check if summary is None or empty and print appropriate message

- Print formatted header: "=== INVENTORY SUMMARY ==="
- Display total batches and total weight with proper formatting
- Print "By Bean Type:" section with weight and percentage for each type
- Calculate percentage: `(weight / summary['total_weight']) * 100`
- Print "By Processing Stage:" section with weight and percentage for each stage
- Use appropriate number formatting (e.g., 1 decimal place)
- Handle division by zero when calculating percentages
- Example output format: "Arabica: 800.0 kg (66.7%)"
- Create library instance: `library = Library("City Public Library", "123 Main St, Anytown")`
- Add initial sample books and members for demonstration
- Implement while loop for menu-driven interface
- Display menu options:
    - Add New Book
    - Add New Member
    - Checkout Book
    - Return Book
    - Display All Books
    - Display All Members
    - Search for Books
    - Exit
- Handle each menu choice with appropriate input prompts and method calls
- Include error handling with try-except blocks for invalid inputs
- For book addition: Prompt for book type (Fiction/Non-Fiction) and create appropriate subclass
- For member addition: Handle email validation errors
- For search: Provide sub-menu for title/author/available books search
- Display formatted results for all operations
- Continue until user chooses to exit

## 6. Execution Steps to follow:

1. Run the program

2. View the main menu

3. Select operations:

- Option 1: Add New Book
- Option 2: Add New Member
- Option 3: Checkout Book
- Option 4: Return Book
- Option 5: Display All Books
- Option 6: Display All Members
- Option 7: Search for Books
- Option 0: Exit

4. Perform operations on the library system

5. View results after each operation

6. Exit program when finished

Execution Steps to Follow:

- All actions like build, compile, running application, running test cases will be through Command Terminal.

- To open the command terminal the test takers, need to go to Application menu (Three horizontal lines at left top) -> Terminal -> New Terminal

- This editor Auto Saves the code

- If you want to exit(logout) and continue the coding later anytime (using Save & Exit option on Assessment Landing Page) then you need to use CTRL+Shift+B -command compulsorily on code IDE. This will push or save the updated contents in the internal git/repository. Else the code will not be available in the next login.

- These are time bound assessments the timer would stop if you logout and while logging in back using the same credentials the timer would resume from the same time it was stopped from the previous logout.

- To setup environment:

- Pip install --upgrade typing_extensions

- To launch application: Python3  filename.py

- To run Test cases: In the terminal type pytest

● Before Final Submission also, you need to use CTRL+Shift+B - command compulsorily on code IDE, before final submission as well. This will push or save the updated contents in the internal git/repository, and will be used to evaluate the code quality.

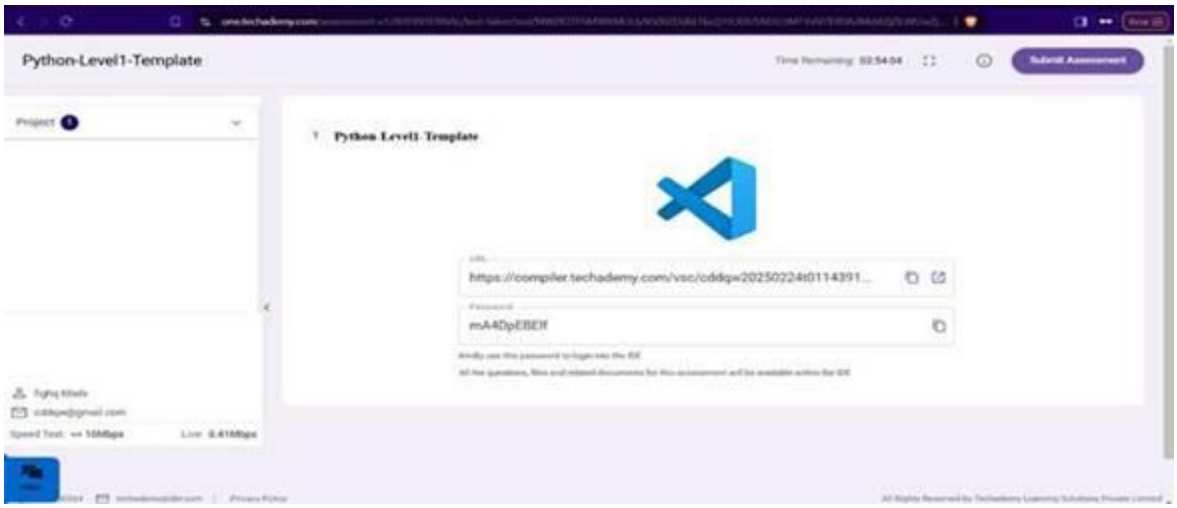
Screen shot to run the program

To run the application

Python3 filename.py

To run the testcase

type pytest in the terminal




● Once you are done with development and ready with submission, you may navigate to the previous tab and submit the workspace. It is mandatory to click on “Submit Assessment” after you are done with code.