**Overview of the System**

**1. Abstract**

This report describes the design and implementation of the NextGen Library Portal, which is a console-based system designed to handle library activities. The system is implemented using an Object-Oriented Programming (OOP) technique to manage main functions including book and member management, borrowing, returning, and fine computation. The design focuses on modularity, data encapsulation, and code reusability to design a stable and maintainable system.

**2. Introduction**

The NextGen Library Portal is a Python-based system that automates library management tasks such as book management, member enrolment, borrowing, returning, and fine calculation. The project applies Object-Oriented Programming (OOP) principles with file persistence to store data.

**3. Requirements**

NextGen Library Portal has to satisfy the following requirements of functionality:

* Book Management
* Member Management
* Borrowing & Return
* Fine Calculation
* Data Persistence
* User Interface

**4.System Design**

**Modules**

os module: It is used to open files and navigate through files.

Datetime module: from datetime import datetime, timedelta is used to import two classes.

1. datetime: This class is used to work with dates and times.

2. timedelta: This class represents a duration

typing: from typing import Dict, List is used to import type hints from Python’s typing module. These help you specify the expected data types for variables, function parameters, and return values, making your code easier to understand and maintain.

**Design Approach**

The project follows an Object-Oriented Programming (OOP) method. This approach structures the system as a group of objects that work together, each representing a real-world entity.

• Encapsulation: Methods and data are bundled together inside classes.

The BaseMember class includes the list of borrowed books (\_\_borrowed) and keeps it hidden from direct access. It can only be changed through special methods like \_add\_loan and \_remove\_loan.

• Inheritance: The system uses a class inheritance structure where StudentMember and FacultyMember come from a common BaseMember class. This helps reuse code and makes it easier to add new member types later without altering the main logic.

• Polymorphism: All members are treated as BaseMember objects by the LibraryPortal class. When a function like loan\_days() or fine\_per\_day () is called, the appropriate implementation for a student or faculty member runs automatically, demonstrating polymorphism.

• Separation of Concerns: The design clearly separates the data models (Book, BaseMember), the main business logic (LibraryPortal), and the user interface (run\_cli). This makes the code easier to maintain, and different parts can be updated independently.

• List Comprehension:

When a book is returned, the borrowed list is filtered in one line using [x for x in self.\_\_borrowed if x["book\_id"] != book\_id]. This removes the returned book without requiring multiple loops.

When saving member data to a file, the borrowed list is serialized using ";".join([f"{x['book\_id']},{x['issue\_date']},{x['due\_date']}" for x in self.\_\_borrowed]). This converts all borrowed book entries into a compact string format.

• UTF-8 Encoding: The project uses encoding="utf-8" for reading and writing files. This guarantees that book titles, author names, and member names are reliably stored in any language. UTF-8 is a universal standard that enhances portability, internationalization, and compatibility.

**A black and white diagram

AI-generated content may be incorrect.**

**Constants**  
• BOOKS\_FILE\_NG :Stores all book details persistently.  
• MEMBERS\_FILE\_NG : Stores all member details persistently.  
• LOAN\_DAYS\_STUDENT : Maximum loan period for students (10 days).  
• LOAN\_DAYS\_FACULTY : Maximum loan period for faculty (21 days).  
• FINE\_STUDENT : Fine for students per late day (Rs.3).  
• FINE\_FACULTY: Fine for faculty per late day (Rs.1).

Usage in Project  
• File Persistence: BOOKS\_FILE\_NG and MEMBERS\_FILE\_NG are used by the LibraryPortal class to load and save data.  
• Loan Rules: LOAN\_DAYS\_STUDENT and LOAN\_DAYS\_FACULTY are used in StudentMember and FacultyMember classes to set due dates.  
• Fine Calculation: FINE\_STUDENT and FINE\_FACULTY are used in return operations to calculate fines for overdue books.  
• Consistency: Instead of hardcoding numbers, constants ensure uniform rules across the project.

**Classes and Methods**

The application is organized around a number of important classes, each responsible for certain tasks:

**4.1 Book Class**  
class Book:  
 def \_\_init\_\_(self, book\_id: str, title: str, author: str, isbn: str, available: bool = True):  
 self.book\_id = book\_id  
 self.title = title  
 self.author = author  
 self.isbn = isbn  
 self.available = available  
  
 def to\_line(self):  
 return DELIM.join([self.book\_id, self.title, self.author, self.isbn, "1" if self.available else "0"])+ "\n"

• to\_line (): Serializes book data into a text line (for saving to file).

**DELIM:**

Each field (like book\_id, title, author, etc.) is clearly separated. The data can be easily parsed or read later by splitting the string using the same delimiter. It maintains consistency in formatting across multiple records.

**join()** takes a list of strings and merges them into one single string, separated by "|"(delimiter).

@staticmethod  
 def from\_line(line: str) -> "Book":  
 p = line.strip().split(DELIM)  
 return Book(p[0], p[1], p[2], p[3], p[4] == "1")

• from\_line(line):

This is a **static method** that takes a single string line (likely from a file or database) and converts it into a Book object. This method belongs to the class but doesn't need access to self or cls. It is used to recreate an object directly from a text line in the file, without needing an existing object

return Book(p[0], p[1], p[2], p[3], p[4] == "1") :

This line reconstructs a Book object from saved file data (ng\_books.txt), converting the availability field "1"/"0" into a proper Boolean (True/False).

Attributes:  
• book\_id: Unique ID for the book.   
• title: Title of the book.  
• author: Author’s name.  
• isbn: International Standard Book Number.  
• available: Boolean value (True if available, False if issued).

Usage in Project:  
• Used by Library Portal to maintain the library’s collection.  
• Provides a way to store and retrieve book data in files.  
• Attribute available helps track whether the book is issued or not.

**4.2 BaseMember Class**

class BaseMember:  
 def \_\_init\_\_(self, member\_id: str, name: str):  
 self.member\_id = member\_id  
 self.name = name  
 self.\_\_borrowed: List[dict] = []

@property  
 def borrowed(self) -> List[dict]:  
 return list(self.\_\_borrowed)

• borrowed (@property): Returns a read-only copy of borrowed books.

It returns a copy (list(self.\_\_borrowed)) to prevent external changes.  
  
 def \_add\_loan(self, book\_id: str, issue\_date: datetime, due\_date: datetime):  
 self.\_\_borrowed.append({  
 "book\_id": book\_id,  
 "issue\_date": issue\_date.isoformat(),  
 "due\_date": due\_date.isoformat(),  
 })

• \_add\_loan(book\_id, issue\_date, due\_date): Adds a borrowed book to the list.

isoformat(): This method converts the datetime object into a string formatted according to the ISO 8601 standard, which looks like:YYYY-MM-DDTHH:MM:SS  
  
 **def \_remove\_loan(self, book\_id: str):  
 self.\_\_borrowed = [x for x in self.\_\_borrowed if x["book\_id"] != book\_id]**

• \_remove\_loan(book\_id): Removes a book from borrowed list.

self.\_\_borrowed = [x for x in self.\_\_borrowed if x["book\_id"] != book\_id] uses a list comprehension to update the member’s borrowed list when a book is returned. It filters out the dictionary entry matching the returned book\_id, ensuring that only the books still borrowed remain in the list. This approach automatically removes the returned book without requiring an loop or manual deletion.  
  
 def loan\_days(self) -> int:  
 return 14

• loan\_days(): Default loan duration (14 days).  
  
 def fine\_per\_day(self) -> int:  
 return 2

• fine\_per\_day() : Default fine (Rs.2/day).  
  
 def display\_info(self) -> str:  
 return f"Member {self.member\_id} - {self.name}"

• display\_info() : Displays member details.  
  
 def to\_line(self) -> str:  
 borrowed\_ser = ";".join([f"{x['book\_id']},{x['issue\_date']},{x['due\_date']}" for x in self.\_\_borrowed])  
 return DELIM.join([self.\_\_class\_\_.\_\_name\_\_, self.member\_id, self.name, borrowed\_ser]) +"\n"

• to\_line() / from\_line()(@static method) : Save and load members from file.

borrowed\_ser = ";".join([f"{x['book\_id']},{x['issue\_date']},{x['due\_date']}" for x in self.\_\_borrowed])

**For each borrowed book stored in the private list \_\_borrowed, take its book\_id, issue\_date, and due\_date.**

return DELIM.join([self.\_\_class\_\_.\_\_name\_\_, self.member\_id, self.name, borrowed\_ser]) +"\n"

The member details are combined into a single text line using the delimiter.

**Self.\_\_class\_\_.name\_\_:** if a method is called on a StudentMember object, Python first checks the StudentMember class. If not found, it moves to the parent BaseMember, and finally to the built-in object class. This mechanism allows overriding, ensuring that student- and faculty-specific rules are applied correctly.

StudentMember → BaseMember → object

FacultyMember → BaseMember → object

@staticmethod  
 def from\_line(line: str) -> "BaseMember":

* the arrow -> indicates a return type annotation

The from\_line() will return a StudentMember object, but still named as a BaseMember type because both StudentMember and FacultyMember inherit from BaseMember

p = line.strip().split(DELIM)  
 mtype, mid, name = p[0], p[1], p[2]  
 borrowed\_ser = p[3] if len(p) > 3 else ""  
 if mtype == "StudentMember":  
 m: BaseMember = StudentMember(mid, name)  
 elif mtype == "FacultyMember":  
 m = FacultyMember(mid, name)  
 else:  
 m = BaseMember(mid, name)  
 if borrowed\_ser:  
 items = borrowed\_ser.split(";")  
 for item in items:  
 if not item:  
 continue  
 bid, issue, due = item.split(",")  
 m.\_add\_loan(bid, datetime.fromisoformat(issue), datetime.fromisoformat(due))  
 return m

Attributes:  
• member\_id: Unique ID of the member.  
• name: Name of the member.  
• \_\_borrowed: Encapsulated list of borrowed books with issue and due dates.  
  
Usage in Project  
• Parent class for all members(Student and Faculty members).  
• Implements encapsulation (\_\_borrowed) and provides core loan handling.  
• Subclasses override loan duration and fine.

**4.3 StudentMember Class**

class StudentMember (BaseMember):  
 def loan\_days(self) -> int:  
 return LOAN\_DAYS\_STUDENT

• loan\_days(): Returns 10 (students can borrow for 10 days).

def fine\_per\_day(self) -> int:  
 return FINE\_STUDENT

• fine\_per\_day() : Returns 3 (Rs.3 fine per overdue day).  
  
 def display\_info(self) -> str:  
 **return f"Student {self.member\_id} - {self.name} (Loan: {self.loan\_days()} days, Fine/day:{self.fine\_per\_day()})"**

• display\_info(): Displays student-specific info.

It returns a statement produces a formatted string that includes the member ID, name, loan duration, and fine rules. It makes sure that when viewing members, students are clearly distinguished from faculty.

Attributes:  
• Inherits all attributes from BaseMember.

Usage in Project  
• Used when a student is enrolled.  
• Ensures that student members follow strict borrowing rules compared to faculty.

**4.4 FacultyMember Class**

class FacultyMember(BaseMember):  
 def loan\_days(self) -> int:  
 return LOAN\_DAYS\_FACULTY

• loan\_days() : Returns 21 (faculty get 21 days).

def fine\_per\_day(self) -> int:  
 return FINE\_FACULTY

• fine\_per\_day() : Returns 1 (Rs.1 fine per overdue day).  
  
 def display\_info(self**) ->** str:  
 return f"Faculty {self.member\_id} - {self.name} (Loan: {self.loan\_days()} days, Fine/day:{self.fine\_per\_day()})"

• display\_info(): Displays faculty-specific info.

Attributes:  
• Inherits all attributes from BaseMember.  
  
Usage in Project:  
• Used when a faculty member is enrolled.  
• Allows longer borrowing duration and lower fines.

**4.5 LibraryPortal Class**

class LibraryPortal:  
 def \_\_init\_\_(self):  
 **self.books: Dict[str, Book] = {}**  
 **self.members: Dict[str, BaseMember] = {}  
 self.\_load()**  
• In the LibraryPortal class, attributes are not passed into the constructor as parameters. Instead, they are directly initialized inside the constructor with default values (self.books = {}, self.members = {}). The type hints indicate the expected data structure but do not assign values.

Immediately after initialization, the \_load() method fills these attributes with data from files. This design ensures that the system always starts with the saved state of books and members, rather than requiring manual input when the object is created.

**def \_load(self):  
 if os.path.exists(BOOKS\_FILE\_NG):  
 with open(BOOKS\_FILE\_NG, "r", encoding="utf-8") as f:  
 for line in f:  
 if line.strip():  
 b = Book.from\_line(line)  
 self.books[b.book\_id] = b**  
 **if os.path.exists(MEMBERS\_FILE\_NG):  
 with open(MEMBERS\_FILE\_NG, "r", encoding="utf-8") as f:  
 for line in f:  
 if line.strip():  
 m = BaseMember.from\_line(line)  
 self.members[m.member\_id] = m**

• if os.path.exists(BOOKS\_FILE\_NG):  
 Checks if the file ng\_books.txt exists.  
Prevents errors if the file is missing (e.g., first time running the program).  
• with open(BOOKS\_FILE\_NG, "r", encoding="utf-8") as f:  
Opens the ng\_books.txt file in read mode with UTF-8 encoding.  
with ensures the file is properly closed after use.

• for line in f:  
 Reads the file line by line. Each line represents a saved book.

• if line.strip():  
Skips blank lines (to avoid errors from empty strings).  
• b = Book.from\_line(line)  
 Converts the text line into a Book object using the from\_line() method.  
• self.books[b.book\_id] = b  
Stores the book object in the self.books dictionary using the book’s ID as the key.• if os.path.exists(MEMBERS\_FILE\_NG):  
Checks if the file ng\_members.txt exists.  
• with open(MEMBERS\_FILE\_NG, "r", encoding="utf-8") as f:  
 Opens the members file for reading.  
• for line in f:  
 Reads each line, where every line stores one member (student/faculty).  
• if line.strip():  
Skips empty lines.  
• m = BaseMember.from\_line(line)  
Reconstructs a member object (StudentMember or FacultyMember) from the line.  
• self.members[m.member\_id] = m

Stores the member object in the self.members dictionary using the member’s ID as the key**.**

**def \_save(self):**• \_save() : Meant to be used internally whenever the system updates data.

with open(BOOKS\_FILE\_NG, "w", encoding="utf-8") as f:  
 for b in self.books.values():  
 f.write(b.to\_line())  
 with open(MEMBERS\_FILE\_NG, "w", encoding="utf-8") as f:  
 for m in self.members.values():  
 f.write(m.to\_line())

def insert\_book(self, book\_id: str, title: str, author: str, isbn: str):  
 if book\_id in self.books:  
 print("Book ID exists")

return

**self.books[book\_id] = Book(book\_id, title, author, isbn)**

• creates a new Book object with the given details and stores it in the self.books dictionary using its book\_id as the key. This allows quick access to each book in the system and ensures that every book can be uniquely identified and managed.

self.\_save()  
 print("Book inserted")  
  
 def discard\_book(self, book\_id: str):  
 b = self.books.get(book\_id) if not b:  
 print("Book not found")  
 return  
 if not b.available:  
 print("Cannot discard issued book")  
 return  
 del self.books[book\_id]  
 self.\_save()  
 print("Book discarded")  
  
 def enroll\_member(self, mtype: str, member\_id: str, name: str):  
 if member\_id in self.members:  
 print("Member ID exists")  
 return  
 **if mtype.lower().startswith("stud"):**

•The input mtype (like "Student" or "STUD") is converted to lowercase.  
 startswith("stud") checks if the input begins with "stud".  
 If true, a StudentMember object is created with the given member\_id and name. m = StudentMember(member\_id, name)  
 **elif mtype.lower().startswith("fac"):**

•If the first condition fails, the program checks whether mtype starts with "fac".  
 If true, a FacultyMember object is created. m = FacultyMember(member\_id, name)  
 else:  
 m = BaseMember(member\_id, name)  
 self.members[member\_id] = m  
 self.\_save()  
 print("Member enrolled")  
  
 def borrow(self, member\_id: str, book\_id: str):  
 m = self.members.get(member\_id)  
 b = self.books.get(book\_id)  
 if not m or not b:  
 print("Invalid member/book")  
 return  
 if not b.available:  
 print("Book not available")  
 return  
 **issue = datetime.now()  
 due = issue + timedelta(days=m.loan\_days())**

• When a book is borrowed, the system records the current date and time as the issue date using datetime.now (). It then calculates the due date by adding the member’s allowed loan period (10 days for students, 21 days for faculty) to the issue date using timedelta. This ensures that each member type automatically receives the correct return deadline.

m.\_add\_loan(book\_id, issue, due)  
 b.available = False  
 self.\_save()  
 **print(f"Issued '{b.title}' to {m.name}. Due: {due.date()}")**

• This line uses an f-string to print a clear and informative message whenever a book is borrowed. It dynamically includes the book title, the member’s name, and the calculated due date, ensuring that users immediately know which book they received and when it must be returned.  
 def receive(self, member\_id: str, book\_id: str):  
 m = self.members.get(member\_id)  
 b = self.books.get(book\_id)  
 if not m or not b:  
 print("Invalid member/book")  
 return

loan = None  
 for x in m.borrowed:  
 **if x["book\_id"] == book\_id:**

• The condition is used to find the exact borrowed record of the book being returned. Since each borrowed book is stored as a dictionary with its book\_id, issue\_date, and due\_date, this check ensures the system matches the correct entry before removing it and calculating any fines.

loan = x  
 break  
 if not loan:  
 print("No active loan found for this book.")  
 return  
 **now = datetime.now()  
 due = datetime.fromisoformat(loan["due\_date"])  
 late\_days = max(0, (now.date() - due.date()).days)  
 fine = late\_days \* self.members[member\_id].fine\_per\_day()**

• When returning a book, the system initially logs the present date (now) and checks it against the saved due date (due). It then determines the number of overdue days by late\_days = max(0, (now.date() - due.date()).days), keeping the value at least zero.

Lastly, the fine is computed by multiplying the late days and the member's rate of daily fine. The process guarantees precise fine calculation in distinguishing between students and staff.

m.\_remove\_loan(book\_id)  
 b.available = True  
 self.\_save()  
 print(f"Returned '{b.title}'. Fine: {fine}")  
  
 def view\_books(self):  
 for b in self.books.values():  
 **print(f"[{b.book\_id}] {b.title} by {b.author} (ISBN {b.isbn}) — {'Available' if b.available else'Issued'}")**

• This f-string prints all details of a book in a clear format, including its ID, title, author, ISBN, and current status. The conditional expression ensures that the status is shown as “Available” if the book is free, or “Issued” if it has been borrowed. This makes the book list easy to read and understand for users. def view\_members(self):  
 for m in self.members.values():  
 print(m.display\_info())  
 if m.borrowed:  
 for loan in m. Borrowed:  
 **print(f" - {loan['book\_id']} (Due {loan['due\_date']})")**

• This f-string is used inside the view\_members() method to display each borrowed book of a member in a list format. It prints the book’s ID along with its due date, making it clear which books are borrowed and when they must be returned.

Attributes:  
• books : Dictionary of all Book objects (book\_id : Book).  
• members : Dictionary of all Member objects (member\_id : Member).  
  
Methods:  
• \_\_init\_\_() : Loads data from files into dictionaries.  
• \_load() / \_save() : Handles persistence of books and members.  
• insert\_book(book\_id, title, author, isbn) : Adds a new book.  
• discard\_book(book\_id) : Removes a book (if not issued).  
• enroll\_member(mtype, member\_id, name) : Enrolls a student or faculty.  
• borrow(member\_id, book\_id) : Issues a book to a member and sets due date.  
• receive(member\_id, book\_id) : Accepts a returned book, calculates fine if overdue.  
• view\_books() : Displays all books with availability status.  
• view\_members() : Displays all members and their borrowed books.

Usage in Project:  
• Acts as the main controller of the library system.  
• Connects books and members to handle borrowing, returning, and fines.  
• Ensures data is always stored in files (ng\_books.txt, ng\_members.txt).

**4.6 CLI (Command-Line Interface)**

def run\_cli():  
 portal = LibraryPortal()  
 MENU = """  
NextGen Library Portal  
1) Insert Book  
2) Discard Book  
3) Enroll Member (Student/Faculty)  
4) Borrow  
5) Receive (Return)  
6) View Books  
7) View Members  
0) Exit  
Choice: """  
 while True:  
 choice = input(MENU).strip()  
 if choice == "1":  
 portal.insert\_book(input("Book ID: "), input("Title: "), input("Author: "), input("ISBN: "))  
 elif choice == "2":  
 portal.discard\_book(input("Book ID: "))  
 elif choice == "3":  
 portal.enroll\_member(input("Type (Student/Faculty): "), input("Member ID: "),input("Name: "))  
 elif choice == "4":  
 portal.borrow(input("Member ID: "), input("Book ID: "))  
 elif choice == "5":  
 portal.receive(input("Member ID: "), input("Book ID: "))  
 elif choice == "6":  
 portal.view\_books()  
 elif choice == "7":  
 portal.view\_members()  
 elif choice == "0":  
 break  
 else:  
 print("Invalid choice.")  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 run\_cli()  
 pass

Explanation:  
The CLI method (run\_cli()) provides a menu-driven interface for the user to perform library operations without directly writing Python code. It acts as the front-end of the system while LibraryPortal is the back-end controller.  
  
Users can:  
• Insert and discard books  
• Enroll members (Student/Faculty)  
• Borrow and return books  
• View all books and members  
• Exit the program CLI (Command-Line Interface)

Usage in Project:  
• The CLI calls the methods of LibraryPortal to perform operations.  
• Provides a user-friendly interface to test the system.  
• Helps demonstrate how the backend logic works without needing a GUI.

• The CLI calls the methods of LibraryPortal to perform operations.  
• Provides a user-friendly interface to test the system.  
• Helps demonstrate how the backend logic works without needing a GUI.

**Conclusion**

The NextGen Library Portal is an organized application that manages library operations effectively. Using OOP concepts, the system works smoothly and is scalable and easy to maintain. It features separate classes for data models and a single LibraryPortal class that contains business logic, which keeps the codebase clean and structured. Member types can be easily customized through inheritance and polymorphism, and file-based storage ensures that all information is securely saved between sessions.