1. Plan for User Interface - NoteApp (Console-Based)

Main Menu:

1. Register
2. Login
3. Add Note
4. View All Notes
5. Exit
6. Delete Note

User Registration:

Enter your choice: 1
Enter new username: [User enters username]
Enter password: [User enters password]

User Login:

Enter your choice: 2
Enter username: [User enters username]
Enter password: [User enters password]

Add Note:

Enter your choice: 3
Enter note title: [User enters note title]
Enter note content: [User enters note content]
Enter note category: [User enters category]

View All Notes:

Enter your choice: 4
[Display a list of all notes with their details]

Delete Note:

Enter your choice: 6
Enter the title of the note you want to delete: [User enters note title]

Exit Application:

Enter your choice: 5

Exiting the NoteApp. Goodbye!

Use Cases:

User Registration:
User selects 'Register.'
User provides a new username and password.

User Login:
User selects 'Login.'
User enters their username and password.

Add Note:
User selects 'Add Note.'
User enters the title, content, and category of the note.

View All Notes:
User selects 'View All Notes.'
The system displays a list of all notes with their details.

Delete Note:
User selects 'Delete Note.'
User enters the title of the note they want to delete.

Exit Application:
User selects 'Exit.'
The system exits the NoteApp.

2. Class Diagram - NoteApp

NoteApp Class:

Purpose: Orchestrates user interactions and manages core functionalities.

Members:
inputHandler: InputHandler
userManager: UserManager
categoryManager: NoteCategoryManager
noteCollection: NoteCollectionWithCategories

Methods: start(): void
main(String[] args): void

InputHandler Class:

Purpose: Handles user input.

Members: None

Methods: getUserInput(String prompt): String

UserManager Class:

Purpose: Manages user registration, login, and authentication.

Members: users: List<User>

Methods: addUser(User user): void
getUserByUsername(String username): User
authenticateUser(User user, String password): boolean

NoteCategoryManager Class:

Purpose: Manages note categories.

Members: categories: List<NoteCategory>

Methods: getOrCreateCategory(String categoryName): NoteCategory

NoteCollectionWithCategories Class:

Purpose: Extends NoteCollection to include categories in the NoteApp application.

Members: None (inherits notes from NoteCollection)

Methods: deleteNoteByTitle(String title): boolean

Note Class:

Purpose: Represents a basic note.

Members: title: String
content: String
Methods: getTitle(): String
getContent(): String
toString(): String

NoteWithCategory Class:

Purpose: Represents a note with an assigned category.

Members: category: NoteCategory

Methods: getCategory(): NoteCategory

NoteCategory Class:

Purpose: Represents a category for notes.

Members: name: String

Methods: getName(): String

NoteCollection Class:

Purpose: Manages a collection of notes.

Members: notes: List<Note>

Methods: addNote(Note note): void
getAllNotes(): List<Note>
removeNote(Note note): void

User Class:

Purpose: Represents a user in the system.

Members: username: String
password: String

Methods: authenticate(String enteredPassword): boolean

3. Java Source Code

User Interface:

```
// This class represents the User Interface of your application.
public class UserInterface {
   // Methods for handling user interactions and displaying information.
}
```

InputHandler:

```
// Handles user input in the console-based application.
public class InputHandler {
    // Methods for getting user input.
}
```

Note:

```
// Represents a basic note in the system.
public class Note {
    private String title;
    private String content;

    public Note(String title, String content) {
        this.title = title;
        this.content = content;
    }

    // Getters and setters for title and content.
}
```

NoteWithCategory:

```
// Represents a note with a category.
public class NoteWithCategory extends Note {
    private NoteCategory category;

    public NoteWithCategory(String title, String content, NoteCategory category) {
        super(title, content);
        this.category = category;
    }

    // Getter and setter for category.
}
```

NoteCategory:

```
// Represents a category for organizing notes.
public class NoteCategory {
    private String categoryName;
    public NoteCategory(String categoryName) {
        this.categoryName = categoryName;
    }

    // Getter for categoryName.
}
```

NoteCollection:

```
// Represents a collection of notes.
public class NoteCollection {
    private List<Note> notes;

    public NoteCollection() {
        this.notes = new ArrayList<>();
    }

    // Methods for adding, getting, and removing notes.
}
```

User:

```
// Represents a user in the system.
public class User {
    private String username;
    private String password;

    public User(String username, String password) {
        this.username = username;
        this.password = password;
    }

    // Method for authenticating the user.
}
```

UserManager:

```
// Manages user-related operations.
public class UserManager {
    private List<User> users;

    public UserManager() {
        this.users = new ArrayList<>();
    }
    // Methods for adding, getting, and authenticating users.
}
```

NoteCategoryManager:

```
// Manages note category-related operations.
public class NoteCategoryManager {
    private List<NoteCategory> categories;

    public NoteCategoryManager() {
        this.categories = new ArrayList<>();
    }

    // Methods for adding, getting, and removing note categories.
}
```

NoteApp:

```
// The main class representing the NoteApp application.
public class NoteApp {
    // Components for user interaction and data management
    private InputHandler inputHandler;
    private UserManager userManager;
    private NoteCategoryManager categoryManager;
    private NoteCollection noteCollection;

    public NoteApp() {
        // Initialize components.
    }

    public void start() {
        // Main application logic and user interactions.
    }
}
```

4. Development Process Summary:

During the development of the NoteApp project, I started by outlining a plan for the user interface and creating a class diagram to define the structure of the application. The initial plan included essential components such as user registration, login, note creation, viewing all notes, and deleting notes. This served as a roadmap for the project and helped in visualizing the interactions between different classes and components.

As I began coding, I followed the principles of abstraction, encapsulation, composition, and inheritance, ensuring that each class had a clear and defined purpose. The classes, including User, Note, NoteWithCategory, NoteCategory, and their managers, were implemented to handle user interactions, note management, and category organization.

However, during the implementation phase, I encountered challenges related to the visibility of the notes list in the NoteCollectionWithCategories class. After addressing this issue, the deleteNoteByTitle method was successfully integrated into the application.

The user interface was designed to operate in a console/command-line environment. The main menu provided options for user registration, login, note creation, viewing all notes, deleting notes, and exiting the application. The interaction flow aligned with the initial plan, offering a straightforward and user-friendly experience.

In retrospect, the development process highlighted the importance of planning and how it guided the coding process. The class diagram was particularly beneficial in visualizing the relationships between classes and refining the structure of the application. However, as with any project, some adjustments were made during implementation to overcome specific challenges.

To improve the project further, I could consider enhancing the error-handling mechanisms, providing more detailed feedback to users, and potentially incorporating additional features to enrich the user experience. Additionally, refining the user interface or considering a graphical user interface (GUI) could be explored to make the application more intuitive and visually appealing.