Develop Notification Service

1. **Due Date Reminders**: Send notifications to users reminding them of due dates for returned books.

2. **Reservation Notifications**: Notify users when a reserved book becomes available.

User Story 1: Send Due Date Reminders

Description:
As a system, I want to send due date reminders to users, so that they are aware of when to return their borrowed books and avoid penalties.

Acceptance Criteria:

1. The system should identify all loaned books with upcoming due dates.
2. The users associated with these books should be notified at least 48 hours before the due date.
3. The notification should include the title of the book and the exact due date.
4. The notification should be sent through the user's preferred channel (e.g., email, SMS).

Below is the detailed design for the "Send Due Date Reminders" user story. This design considers various components like entities, controllers, services, DAO, and interactions with other microservices.

Entities
1. **DueDateReminder**: This entity would represent a single due date reminder.
   - Fields: `id`, `userId`, `bookId`, `dueDate`, `sent`, `channel`

Controllers
1. **DueDateReminderController**: This controller will expose endpoints for internal use.
   - `GET /reminders/upcoming`: Get a list of reminders that are yet to be sent.
   - `POST /reminders/mark-sent`: Mark a reminder as sent.

Services
1. **DueDateReminderService**: Service for business logic.
   - `fetchUpcomingReminders()`: Fetch all reminders that are to be sent within 48 hours.

- `markReminderAsSent(id)`: Mark a reminder as sent.
- `sendDueDateReminders()`: Business logic to send reminders, called periodically.

DAO
1. **DueDateReminderDAO**: Data access object for `DueDateReminder`.
   - `findUpcomingReminders()`: Finds all reminders with `sent=false` and `dueDate` within 48 hours.
   - `markAsSent(id)`: Updates the `sent` status to `true` for a given reminder.

Integration to other MS
1. **Loan Service**: Will be used to fetch loan records and due dates.
2. **User Service**: Will be used to fetch the user's preferred communication channel and their contact details.

Other Considerations
1. **Scheduler**: A scheduler will run every hour to trigger the `sendDueDateReminders()` method in the service layer.
2. **Notification Provider**: Implementing a separate class to handle sending notifications via different channels (SMS, Email).
3. **Logging**: All sending failures and successes should be logged for auditing purposes.

Sample Flow:

1. The Scheduler triggers `sendDueDateReminders()` in the service layer.
2. `fetchUpcomingReminders()` gets called, which in turn uses `DueDateReminderDAO` to fetch reminders that need to be sent.
3. For each reminder, the service will:
   - Fetch the loan details from Loan Service.
   - Fetch the user details from User Service.
   - Use the Notification Provider to send the notification.
4. Mark the reminder as sent by calling `markReminderAsSent(id)`.

This should cover most of the detailed design for this user story.

User Story 2: Send Reservation Notifications

Description:
As a system, I want to notify users when a reserved book becomes available, so that they can proceed to borrow it.

Acceptance Criteria:

1. The system should identify when a reserved book becomes available.
2. The user who has reserved the book should be immediately notified.
3. The notification should include the title of the book and instructions on how to proceed with borrowing.
4. The notification should be sent through the user's preferred channel (e.g., email, SMS).

Certainly, here's the detailed design for the "Send Reservation Notifications" user story:

Entities

1. **ReservationNotification**: This entity would represent a single reservation notification.
   - Fields: `id`, `userId`, `bookId`, `sent`, `channel`

Controllers

1. **ReservationNotificationController**: This controller will expose endpoints for internal use.
   - `POST /notification/reservation`: Trigger a notification for a reservation becoming available.
   - `POST /notification/mark-sent`: Mark a notification as sent.

Services

1. **ReservationNotificationService**: Service for business logic.
   - `sendReservationNotification(userId, bookId)`: Sends a notification to the user that a reserved book has become available.
   - `markNotificationAsSent(id)`: Mark a notification as sent.

DAO

1. **ReservationNotificationDAO**: Data access object for `ReservationNotification`.
   - `createNotification(userId, bookId)`: Creates a new reservation notification entry with `sent=false`.
   - `markAsSent(id)`: Updates the `sent` status to `true` for a given notification.

Integration to other MS

1. **Book Catalog Service**: To fetch details about the book that has become available.
2. **User Service**: Will be used to fetch the user's preferred communication channel and their contact details.

Other Considerations

1. **Event-Driven Architecture**: This feature could be event-driven, where the Book Catalog Service emits an event whenever a reserved book becomes available, and the Notification Service listens to this event.
2. **Notification Provider**: Implementing a separate class to handle sending notifications via different channels (SMS, Email).
3. **Logging**: All sending failures and successes should be logged for auditing purposes.

Sample Flow:

1. The Book Catalog Service emits an event when a reserved book becomes available.
2. The Notification Service listens to this event and triggers `sendReservationNotification(userId, bookId)` in the service layer.
3. A new notification record gets created using `createNotification(userId, bookId)` via `ReservationNotificationDAO`.
4. The service layer fetches the book details from the Book Catalog Service.
5. The service layer fetches the user's details and preferred notification channel from the User Service.
6. Use the Notification Provider to send the notification.
7. Mark the notification as sent by calling `markNotificationAsSent(id)`.

This design should provide a comprehensive way to implement this user story.