ClubConnect Implementation Guide

Quick Start Commands for Cursor

Use these prompts in Cursor to generate the remaining files:

1. Create Remaining Model Classes

```
Create ClubMember.java model class in package com.clubconnect.models with:
- Fields: membershipId, clubId, userId, role (LEADER/MEMBER enum), status (PENDING/ACTIVE/ALUMNI enum),
joinedAt timestamp
- Constructor, getters, setters, toString method
- Follow the same pattern as User.java
```

```
Create RSVP.java model class with:
- Fields: rsvpId, eventId, userId, status (ATTENDING/NOT_ATTENDING/WAITLIST enum), attended boolean,
rsvpDate
- Constructor, getters, setters
```

```
Create Resource.java model class with:
- Fields: resourceId, resourceName, resourceType, capacity, status (AVAILABLE/UNAVAILABLE)
```

```
Create BudgetTransaction.java with:
- Fields: transactionId, clubId, eventId, transactionType (INCOME/EXPENSE), amount (BigDecimal), category,
description, transactionDate, approvedBy
```

```
Create Announcement.java with:
- Fields: announcementId, clubId, title, content, createdBy, createdAt
```

```
Create Discussion.java with:
- Fields: discussionId, clubId, eventId, userId, parentId (for threading), content, createdAt
```

```
Create Notification.java with:
- Fields: notificationId, userId, title, message, notificationType, isRead, createdAt
```

2. Create DAO Classes

For each model, create a DAO following this pattern:

```
Create ClubDAO.java in package com.clubconnect.dao with methods:
- createClub(Club club): boolean
- getClubById(int clubId): Club
- getAllClubs(): List<Club>
- getClubsByStatus(ClubStatus status): List<Club>
- searchClubs(String keyword): List<Club>
- updateClub(Club club): boolean
- deleteClub(int clubId): boolean
- approveClub(int clubId): boolean
- getClubsByCategory(String category): List<Club>
Follow UserDAO.java pattern using PreparedStatements
```

Repeat similar prompts for:

- EventDAO.java
- ClubMemberDAO.java
- RSVPDAO.java
- ResourceDAO.java
- BudgetTransactionDAO.java
- AnnouncementDAO.java
- DiscussionDAO.java
- NotificationDAO.java

3. Create Service Classes

```
Create NotificationService.java in package com.clubconnect.services with:
- sendNotification(int userId, String title, String message): void
- sendBatchNotifications(List<Integer> userIds, String title, String message): void using multi-threading
- getUnreadNotifications(int userId): List<Notification>
- markAsRead(int notificationId): boolean
Use Thread or ExecutorService for batch operations
Include EmailSender for email notifications
```

Create ClubService.java with:
- createClubRequest(Club club, int userId): boolean
- approveClub(int clubId, int adminId): boolean
- updateClubDetails(Club club): boolean
- searchClubs(String keyword, String category): List<Club>
- getClubMembers(int clubId): List<User>
- getClubStatistics(int clubId): Map<String, Object>
Use ClubDAO and related DAOs

Create EventService.java with:
- createEvent(Event event, int resourceId): boolean (check resource availability)
- updateEvent(Event event): boolean
- cancelEvent(int eventId): boolean (notify all RSVPs)
- checkResourceConflicts(int resourceId, Date date, Time start, Time end): boolean
- getUpcomingEvents(int clubId): List<Event>
- recordAttendance(int eventId, List<Integer> attendedUserIds): boolean
Include background thread for event reminders

Create MembershipService.java with:
- applyToClub(int clubId, int userId): boolean
- approveMembership(int membershipId, int leaderId): boolean
- rejectMembership(int membershipId, int leaderId, String reason): boolean
- promoteMember(int membershipId, int adminId): boolean
- getMembershipStatus(int userId, int clubId): String
- exportMemberList(int clubId, String filename): void (CSV)

Create BudgetService.java with:
- requestBudget(int clubId, int eventId, BigDecimal amount, String description): boolean
- approveBudgetRequest(int transactionId, int adminId): boolean
- recordTransaction(BudgetTransaction transaction): boolean
- getClubBudgetSummary(int clubId): Map<String, BigDecimal>
- getBudgetHistory(int clubId): List<BudgetTransaction>
- checkLowBudget(int clubId): boolean (for automated alerts)

```
Create ReportService.java with:
- generateAttendanceReport(int eventId): List<Map<String, Object>>
- generateClubEngagementReport(int clubId): Map<String, Object>
- generateBudgetReport(int clubId): Map<String, Object>
- exportAttendancePDF(int eventId, String filename): void (using iText)
- exportFinancialReportPDF(int clubId, String filename): void
Include chart data generation for UI visualization
```

4. Create Dashboard Classes

```
Create AdminDashboard.java in package com.clubconnect.ui extending JFrame with:
- Main menu bar with: Clubs, Events, Users, Budgets, Reports, Logout
- Tabbed pane with panels for: All Clubs, Pending Approvals, User Management, Budget Requests, Global Reports
- Search functionality for members (displaying details and associated clubs)
- Implement logout that returns to LoginFrame
Use JTabbedPane, JMenuBar, and custom panels
```

```
Create LeaderDashboard.java extending JFrame with:
- Menu: My Club, Events, Members, Budget, Announcements, Reports
- Panels for: Club details editor, Event calendar, Membership applications, Budget tracker
- Create Event button (opens CreateEventDialog)
- Send Announcement button (multi-threaded sending)
- View other clubs (read-only)
```

```
Create MemberDashboard.java extending JFrame with:
- Menu: Browse Clubs, My Clubs, Events, Profile
- Club directory with search and filter
- Event calendar with RSVP buttons
- My memberships panel
- Discussion board access
```

```
Create GuestDashboard.java extending JFrame with:
- Read-only view of public clubs
- Public event calendar
- Login/Register buttons prominent
- No interaction features (grayed out)
```

5. Create Panel Classes

Create ClubListPanel.java in package com.clubconnect.ui.panels extending JPanel with:
- JTable displaying clubs (name, category, members count, status)
- JTextField for search
- JComboBox for category filter
- Buttons: View Details, Join/Edit (based on role)
- Implement TableModel for Club data

Create EventCalendarPanel.java extending JPanel with:
- Monthly calendar view using JTable or custom painting
- Color coding for event types (MEETING, SEATING, EVENT)
- Click event to show details
- Filter by club
- Upcoming events list sidebar

Create MembershipPanel.java extending JPanel with:
- JTable for membership applications
- Columns: Student Name, Student ID, Applied Date, Status
- Buttons: Approve, Reject (for leaders)
- Export to CSV button

Create BudgetPanel.java extending JPanel with:
- Current budget display (large, prominent)
- JTable for transaction history
- Add transaction button
- Budget allocation by category (pie chart using custom painting)
- Low budget warning label

Create AnnouncementPanel.java extending JPanel with:
- JTextArea for composing announcements
- Send to: All Members checkbox
- Previous announcements list with JList
- Send button (triggers multi-threaded NotificationService)

```
Create ReportPanel.java extending JPanel with:
- Report type selection: Attendance, Engagement, Budget
- Date range pickers
- Generate button
- Results display area (JTable or JTextArea)
- Export to PDF/CSV buttons
- Simple bar chart visualization using Java2D
```

6. Create Dialog Classes

```
Create CreateClubDialog.java in package com.clubconnect.ui.dialogs extending JDialog with:
- Form fields: Club Name, Category (JComboBox), Mission Statement (JTextArea), Initial Budget
- Validation on submit
- Submit button calls ClubService.createClubRequest()
- Modal dialog (setModal(true))
```

```
Create CreateEventDialog.java extending JDialog with:
- Fields: Title, Description, Date (JDateChooser or JTextField), Start Time, End Time
- Venue/Room selection (JComboBox from available resources)
- Capacity, Event Type selection
- Checkbox: Request Budget (shows budget fields when checked)
- Resource conflict checking on date/time selection
- Submit calls EventService.createEvent()
```

```
Create AttendanceDialog.java extending JDialog with:
- JTable with columns: Student Name, Student ID, Present (JCheckBox)
- Prepopulated with RSVPs from event
- Select All / Deselect All buttons
- Save button updates database via EventService
```

```
Create BudgetRequestDialog.java extending JDialog with:
- Event selection (if multiple events need budget)
- Amount field (JTextField with currency validation)
- Description/Justification (JTextArea)
- Category selection
- Submit calls BudgetService.requestBudget()
```

```
Create MemberSearchDialog.java extending JDialog (Admin only) with:
- Search field (name, student ID, email)
- Results JTable showing: Name, Student ID, Email, Role
- View Details button shows user info and associated clubs
- Promote to Leader button (for admins)
```

7. Create Utility Classes

```
Create EmailSender.java in package com.clubconnect.utils with:
- sendEmail(String to, String subject, String body): boolean
- sendBatchEmails(List<String> recipients, String subject, String body): void using threads
Use JavaMail API
Configuration: Gmail SMTP or university email server
```

```
Create CSVExporter.java with:
- exportMembers(List<User> members, String filename): void
- exportEvents(List<Event> events, String filename): void
- exportTransactions(List<BudgetTransaction> transactions, String filename): void
Use FileWriter and proper CSV formatting
```

```
Create PDFGenerator.java with:
- generateAttendanceReport(int eventId, String filename): void
- generateFinancialReport(int clubId, String filename): void
Use iText library for PDF generation
Include tables, headers, and basic formatting
```

Implementation Order

Follow this order for best results:

1. **Models** (1-2 hours)
   - Complete all model classes first
   - Test by creating objects in main method
2. **DAOs** (3-4 hours)
   - Implement one DAO at a time
   - Test each with simple main method before moving on
3. **Services** (4-5 hours)
   - Start with simpler services (ClubService, MembershipService)

- End with complex ones (NotificationService with threading, ReportService)

4. **Utilities** (2-3 hours)
   - Start with CSVExporter (simplest)
   - EmailSender next
   - PDFGenerator last
5. **Panels** (4-5 hours)
   - Create simple panels first (ClubListPanel)
   - More complex panels later (EventCalendarPanel, ReportPanel)
6. **Dialogs** (3-4 hours)
   - Simple forms first (CreateClubDialog)
   - Complex ones later (CreateEventDialog with resource checking)
7. **Dashboards** (5-6 hours)
   - Start with GuestDashboard (simplest)
   - Then MemberDashboard
   - LeaderDashboard
   - AdminDashboard (most complex)
8. **Integration Testing** (2-3 hours)
   - Test complete workflows
   - Test multi-threading
   - Test error scenarios

Total Estimated Time: 25-35 hours

Testing Each Component

Test Template

```java
```

```
public static void main(String[] args) {
    // Initialize database
    DatabaseManager.initializeDatabase();

    // Test your component
    YourDAO dao = new YourDAO();

    // Test create
    YourModel obj = new YourModel(...);
    boolean result = dao.create(obj);
    System.out.println("Create: " + result);

    // Test read
    YourModel retrieved = dao.getById(1);
    System.out.println("Retrieved: " + retrieved);

    // Close
    DatabaseManager.closeConnection();
}
```

Common Cursor Prompts

For any DAO:

```
Create [Entity]DAO.java following the pattern of UserDAO.java with full CRUD operations:
- create[Entity]([Entity] object)
- get[Entity]ById(int id)
- getAll[Entities]()
- update[Entity]([Entity] object)
- delete[Entity](int id)
Include proper exception handling and PreparedStatements
```

For any Service:

```
Create [Feature]Service.java that uses [Entity]DAO and implements business logic for:
- [List main operations]
Include validation, error handling, and any required multi-threading
```

For any UI Component:

```
Create [Component]Frame/Panel/Dialog.java extending J[Frame/Panel/Dialog] with:
- [Describe the UI elements needed]
- [Describe the actions/buttons]
- Use GridBagLayout or BorderLayout
- Include proper event handlers
- Show JOptionPane for feedback
```

Pro Tips for Cursor

1. **Be specific about package names** - Always include full package path
2. **Reference existing files** - "Follow the pattern of UserDAO.java"
3. **List all methods needed** - Don't let Cursor decide what methods you need
4. **Specify libraries** - "Use PreparedStatement", "Use JTable", etc.
5. **Request comments** - Add "Include Javadoc comments" to prompts
6. **One file at a time** - Don't ask for multiple complex files at once
7. **Test incrementally** - Generate, test, fix, then move to next file

Debugging Checklist

- [ ] All imports are correct
- [ ] Package names match directory structure
- [ ] Database connection is established before use
- [ ] PreparedStatements are used (no string concatenation)
- [ ] Resources are closed (use try-with-resources)
- [ ] Exceptions are caught and handled
- [ ] UI components are added to container
- [ ] Event listeners are attached
- [ ] SwingUtilities.invokeLater() used for UI updates from threads
- [ ] Null checks before using objects
- [ ] Input validation before database operations

Good luck with your implementation! The files I've provided give you a solid foundation. Use this guide with Cursor to complete the rest efficiently.