iClub Website - Complete Development Roadmap

Overview

You're building a club management system with three main parts:

1. **Backend (API + Database)** - The engine that stores and manages all data
2. **Admin Portal** - Where authorized people manage content, members, KPIs
3. **Public Website** - What visitors see (achievements, board members, contacts, etc.)

Development Order: Backend → Admin Portal → Public Website

PHASE 1: PLANNING & SETUP (Week 1)

Goal: Understand what you're building before writing code

Task 1.1: Database Schema Design

What to do:

- Get a piece of paper or use draw.io
- List all the "things" you need to store (these become tables)
- For each thing, list what information you need to track (these become fields/columns)

Tables you probably need:

- Members (club members info)
- Users (admin login accounts)

- Achievements (club accomplishments)
- Board_Members (leadership positions)
- Partners (partner organizations)
- KPIs (performance metrics)
- Teams (marketing, HR, technical, etc.)
- Events (if tracking events)

For each table, define:

- What fields/columns does it need?
- What type is each field? (text, number, date, true/false)
- How does it relate to other tables?

Example for Members table:

- id (number, auto-generated)
- fullName (text)
- email (text, must be unique)
- studentId (text)
- department (text)
- yearOfStudy (number)
- phoneNumber (text)
- role (text: "member", "board_member", "alumni")
- teamId (number, links to Teams table)
- isActive (true/false)
- joinDate (date)
- profilePhotoUrl (text, optional)

Deliverable: A document or diagram showing all tables and their fields

Time: 2-3 days of thinking and planning

Task 1.2: API Endpoints Planning

What to do:

- For each table, list what operations admins need to perform
- This determines what API endpoints you'll build

Example for Members:

- View all members → GET /api/members
- View one member → GET /api/members/:id
- Add new member → POST /api/members
- Update member → PUT /api/members/:id
- Delete member → DELETE /api/members/:id
- Filter by team → GET /api/members?teamId=X
- Filter active only → GET /api/members?active=true

Repeat for each table

Deliverable: List of all endpoints you'll need to build

Time: 1 day

Task 1.3: Admin Portal Screens Planning

What to do:

- Sketch what screens your admin portal needs
- What can each admin role do?

Screens needed:

- Login page
- Dashboard (overview of key stats)
- Members management (list, add, edit, delete)
- Achievements management
- Board members management
- Partners management
- KPIs management (probably most complex)
- Teams management
- User accounts management (who can log into admin portal)
- Settings/profile page

For KPIs screen, think through:

- How do different department heads see only their KPIs?
- How is progress tracked?
- What visualizations do you need? (charts, progress bars)

Deliverable: Rough sketches or wireframes of each screen

Time: 1-2 days

PHASE 2: BACKEND DEVELOPMENT (Week 2-4)

Goal: Build the API that will power everything

Task 2.1: Project Setup (COMPLETED ✓)

You already did this! You have:

- Node.js installed
- PostgreSQL installed
- Backend project initialized
- Prisma configured
- First table created
- First endpoint working

Task 2.2: Create All Database Tables

What to do: For each table you planned in Phase 1, you'll:

Step by step for EACH table:

1. **Add model to schema.prisma**
    - Open `prisma/schema.prisma`
    - Add a new model (like you did for Member)
    - Define all fields and their types
    - Define relationships to other tables

2. **Create migration**

```
npx prisma migrate dev --name add_[table_name]_table
```

- This creates the table in your database
- Creates a migration file to track the change

3. **Generate Prisma Client**

```
npx prisma generate
```

- Updates the code that lets you use this table

4. **Verify in pgAdmin**
    - Open pgAdmin
    - Check that the new table exists
    - Look at its structure

Do this for each table one by one

Deliverable: Complete database with all tables created

Time: 3-5 days (depending on number of tables)

Task 2.3: Build API Endpoints - Authentication First

What to do: Before building endpoints for data, you need login/authentication

Sub-task 2.3.1: Create Users table

- This stores admin account info (email, hashed password, role)
- Add to schema.prisma
- Run migration

Sub-task 2.3.2: Install authentication packages

```
npm install bcryptjs jsonwebtoken
npm install --save-dev @types/bcryptjs @types/jsonwebtoken
```

- bcryptjs: encrypts passwords
- jsonwebtoken: creates login tokens

Sub-task 2.3.3: Create authentication endpoints Build these in server.js:

- POST /api/auth/register - Create new admin account
- POST /api/auth/login - Login and get token
- GET /api/auth/me - Get current user info (requires token)

Sub-task 2.3.4: Create middleware for protecting routes

- Build function that checks if request has valid token
- This protects other endpoints (only logged-in users can access)

Deliverable: Working login system

Time: 2-3 days

Task 2.4: Build API Endpoints - Members

What to do: Create all CRUD operations for Members

In server.js, add these endpoints:

1. **GET /api/members** - Get all members
   - Use: `prisma.member.findMany()`
   - Add filters if query params exist (teamId, active, etc.)
2. **GET /api/members/:id** - Get one member
   - Use: `prisma.member.findUnique({ where: { id } })`
3. **POST /api/members** - Create new member
   - Get data from request body
   - Validate required fields
   - Use: `prisma.member.create({ data })`
4. **PUT /api/members/:id** - Update member
   - Use: `prisma.member.update({ where: { id }, data })`
5. **DELETE /api/members/:id** - Delete member
   - Use: `prisma.member.delete({ where: { id } })`

For each endpoint:

- Add authentication check (must be logged in)
- Add permission check (does user have rights to do this?)
- Add validation (is data correct?)
- Add error handling (what if something fails?)

- Test with browser or Postman

Deliverable: Complete Members API working

Time: 2-3 days

Task 2.5: Build API Endpoints - Repeat for Other Tables

What to do: Repeat Task 2.4 for each remaining table:

- Achievements
- Board Members
- Partners
- Teams
- KPIs (this one will be more complex)
- Events (if needed)

For KPIs specifically:

- Add endpoints for filtering by team
- Add endpoints for filtering by time period
- Add endpoint for calculating progress percentages
- Add endpoint for dashboard statistics

Deliverable: Complete API for entire system

Time: 1-2 weeks (depending on complexity)

Task 2.6: Test All Endpoints Thoroughly

What to do:

- Install Postman (free tool for testing APIs)
- Test EVERY endpoint you created
- Try to break them (send wrong data, missing fields, etc.)
- Fix any bugs you find
- Document what each endpoint expects and returns

Create a collection in Postman with:

- All your endpoints
- Example requests for each
- Expected responses

Deliverable: Fully tested, working backend API

Time: 2-3 days

PHASE 3: ADMIN PORTAL DEVELOPMENT (Week 5-8)

Goal: Build the interface where admins manage everything

Task 3.1: React Project Setup

What to do:

1. **Navigate to frontend folder**

```
cd ../frontend
```

2. **Create React app**

```
npx create-react-app admin-portal
cd admin-portal
```

3. **Install necessary packages**

```
npm install react-router-dom axios
npm install @mui/material @emotion/react @emotion/styled
npm install recharts
```

- react-router-dom: for navigation between pages
- axios: for making API calls to your backend
- @mui/material: UI components (forms, tables, buttons)
- recharts: for KPI charts/graphs

4. **Start development server**

```
npm start
```

- Opens browser at localhost:3000
- Shows default React page

Deliverable: React app running

Time: 1 hour

Task 3.2: Setup Routing and Layout

What to do:

1. **Create folder structure**

```
src/
├── components/     (reusable pieces)
├── pages/          (full screens)
├── services/       (API calling functions)
├── context/        (authentication state)
└── utils/          (helper functions)
```

2. **Create basic routes**
    - Login page
    - Dashboard (main page after login)
    - Members page
    - Achievements page
    - Board Members page
    - Partners page
    - KPIs page
    - Teams page
    - Settings page
3. **Create navigation layout**

- Sidebar with links to all pages
- Header with user info and logout button
- Main content area

Deliverable: App structure with empty pages and navigation

Time: 1-2 days

Task 3.3: Build Authentication Flow

What to do:

1. **Create Login Page**
   - Email input field
   - Password input field
   - Login button
   - Error message display
2. **Create authentication service**
   - Function to call POST /api/auth/login
   - Store received token in localStorage
   - Include token in all subsequent API calls
3. **Create protected route wrapper**
   - Checks if user is logged in
   - Redirects to login if not
   - Wraps all other pages
4. **Create logout functionality**

- Clear token from localStorage
- Redirect to login

Deliverable: Working login/logout system

Time: 2-3 days

Task 3.4: Build Members Management Page

What to do: This is your first complete CRUD interface. Pattern repeats for other pages.

Sub-task 3.4.1: Display members list

- Create table component
- Fetch data from GET /api/members when page loads
- Display in table with columns: Name, Email, Department, Role, Status
- Add action buttons (Edit, Delete) for each row
- Add loading state while fetching
- Add error state if fetch fails

Sub-task 3.4.2: Add member form

- Create modal/dialog with form
- Fields for all member data
- "Add Member" button opens form
- Validate inputs (email format, required fields)
- On submit, call POST /api/members
- Close form and refresh list on success

- Show error message on failure

Sub-task 3.4.3: Edit member functionality

- Clicking Edit button opens same form
- Pre-fill with existing member data
- On submit, call PUT /api/members/:id
- Refresh list on success

Sub-task 3.4.4: Delete member functionality

- Clicking Delete shows confirmation dialog
- On confirm, call DELETE /api/members/:id
- Refresh list on success

Sub-task 3.4.5: Add filters/search

- Search box to filter by name or email
- Dropdown to filter by team
- Dropdown to filter by active/inactive

Deliverable: Complete members management interface

Time: 4-5 days

Task 3.5: Build Achievements Management Page

What to do: Same pattern as members, but simpler

Features needed:

- List all achievements (with date, title, description)
- Add new achievement (with image upload)
- Edit achievement
- Delete achievement
- Sort by date

For image upload:

- You'll need image hosting (can use Cloudinary free tier)
- Or store base64 in database (not recommended for many images)
- Or use local storage and serve via backend

Deliverable: Complete achievements management

Time: 2-3 days

Task 3.6: Build Other Management Pages

What to do: Repeat the pattern for:

- Board Members (similar to members, but might include position, term dates)
- Partners (name, logo, website, description)
- Teams (team name, description, head)

Each page needs:

- List view

- Add form
- Edit form
- Delete functionality
- Appropriate filters

Deliverable: All basic management pages working

Time: 1 week

Task 3.7: Build KPIs Management Page (Most Complex)

What to do: This requires more thought based on your KPI structure

Features needed:

1. **Dashboard view**
    - Show all KPIs for user's teams
    - Progress bars or charts
    - Color coding (green = on track, yellow = behind, red = critical)
2. **KPI list by team**
    - Filter by team
    - Filter by time period (monthly, quarterly, yearly)
3. **Add/Edit KPI**
    - KPI title
    - Target value
    - Current value
    - Responsible person (dropdown of members)

- Team (dropdown)
- Time period
- Status

4. **Update progress**
    - Quick update current value
    - Add notes/comments
    - Mark as complete
5. **Visualizations**
    - Charts showing progress over time
    - Team comparison charts
    - Export reports

This is the most custom part - design based on your club's needs

Deliverable: Working KPI management system

Time: 1-2 weeks

Task 3.8: Build Dashboard (Overview Page)

What to do: Create main dashboard with key stats

Widgets to include:

- Total members count
- Active members count
- Recent achievements (last 5)

- KPIs summary (how many on track vs behind)
- Upcoming events (if you have events)
- Quick actions (Add Member, Add Achievement buttons)

Use charts/graphs for visual appeal

Deliverable: Informative dashboard

Time: 2-3 days

Task 3.9: Build User Management Page (Admin Accounts)

What to do: Super Admin can manage who has access to admin portal

Features:

- List all admin users
- Add new admin user (email, role)
- Assign roles (Super Admin, HR Admin, Team Head)
- Deactivate users
- Reset passwords

Deliverable: Admin user management

Time: 2-3 days

Task 3.10: Polish and Improve UX

What to do: Make everything user-friendly

Improvements:

- Add confirmation dialogs for destructive actions
- Add success/error toast notifications
- Add loading spinners
- Improve form validation messages
- Add help text/tooltips
- Make responsive (works on tablets)
- Add keyboard shortcuts
- Improve error handling

Deliverable: Polished, professional admin portal

Time: 3-5 days

PHASE 4: PUBLIC WEBSITE DEVELOPMENT (Week 9-10)

Goal: Build what visitors see

Task 4.1: Plan Public Pages

What pages do you need?

- Home (hero section, overview)
- About Us
- Board Members
- Achievements
- Partners

- Events (if applicable)
- Contact Us
- Join Us / Recruitment

Deliverable: List of pages with content plan

Time: 1 day

Task 4.2: Create Public Website React Project

What to do:

1. **Create new React app**

```
cd ../../frontend
npx create-react-app public-website
cd public-website
```

2. **Install packages**

```
npm install react-router-dom axios
```

3. **Setup routing for all pages**

Deliverable: Project structure with empty pages

Time: 1 day

Task 4.3: Build Each Public Page

What to do: For each page, fetch relevant data from your backend API and display nicely

Home Page:

- Hero section with club mission/vision
- Featured achievements (fetch latest 3 from API)
- Stats (total members, years active, events held)
- Call to action (Join Us button)

About Us:

- Club history
- Mission and vision
- What we do
- Static content (stored in backend or hardcoded)

Board Members:

- Fetch from GET /api/board-members
- Display with photos, names, positions
- Grid or card layout

Achievements:

- Fetch from GET /api/achievements
- Display chronologically or by category
- Include images

Partners:

- Fetch from GET /api/partners
- Display logos in grid
- Link to partner websites

Contact:

- Contact form (sends email or stores in database)
- Social media links
- Location/meeting info

Deliverable: All public pages with real data from backend

Time: 1 week

Task 4.4: Design and Styling

What to do: Make it look professional

Options:

- Use Tailwind CSS for styling
- Use Material-UI for components
- Find a free template to adapt
- Hire a designer (if budget allows)

Focus on:

- Consistent color scheme (club colors)
- Clean, modern design
- Good typography
- Responsive (works on phones)
- Fast loading (optimize images)

Deliverable: Beautiful public website

Time: 3-5 days

PHASE 5: DEPLOYMENT (Week 11)

Goal: Put everything online

Task 5.1: Prepare Backend for Deployment

What to do:

1. **Environment variables**
   - Make sure all sensitive data (database password, JWT secret) is in .env
   - Don't commit .env to GitHub
2. **Add CORS**

```
npm install cors
```

- Configure to allow frontend domains to access API

3. **Add production database URL**
    - Railway will provide this
4. **Test locally one more time**

Deliverable: Backend ready for production

Time: 1 day

Task 5.2: Deploy Backend to Railway

What to do:

1. **Push code to GitHub**
    - Create repository
    - Push your backend code
2. **Create Railway account**
    - Go to railway.app
    - Sign up with GitHub
3. **Create new project**
    - Add PostgreSQL service
    - Add Node.js service (connect to GitHub repo)
4. **Configure environment variables in Railway**
    - DATABASE_URL (Railway provides)
    - JWT_SECRET
    - Any other secrets

5. **Deploy**
    - Railway automatically deploys from GitHub
    - Note the URL Railway gives you
6. **Run migrations on production database**
    - Railway can run this automatically
7. **Test all endpoints**
    - Use Postman with production URL

Deliverable: Backend running on Railway

Time: 1-2 days

Task 5.3: Deploy Admin Portal to Vercel

What to do:

1. **Update API URLs in admin portal**
    - Change from localhost:3000 to Railway URL
    - Use environment variables
2. **Push admin portal to GitHub**
3. **Create Vercel account**
    - Go to vercel.com
    - Sign up with GitHub
4. **Import project**
    - Select admin portal repo
    - Configure build settings (should auto-detect React)

5. **Add environment variables**
    - API_URL = your Railway backend URL
6. **Deploy**
    - Vercel builds and deploys
    - Gives you a URL like: iclub-admin.vercel.app
7. **Test everything**
    - Login, add data, check all features work

Deliverable: Admin portal live online

Time: 1 day

Task 5.4: Deploy Public Website to Vercel

What to do: Same process as admin portal

1. Push to GitHub
2. Import to Vercel
3. Configure environment variables
4. Deploy
5. Test

Deliverable: Public website live online

Time: 1 day

Task 5.5: Custom Domain (Optional)

What to do: If you buy a domain:

1. **Buy domain**
    - Namecheap, Porkbun, etc.
2. **Add to Vercel**
    - In Vercel project settings → Domains
    - Add your custom domain
    - Follow DNS configuration instructions
3. **Configure subdomains**
    - admin.iclubmedasu.com → admin portal
    - www.iclubmedasu.com → public website
    - api.iclubmedasu.com → Railway backend (configure in Railway)

Deliverable: Custom domains working

Time: 1 day (mostly waiting for DNS propagation)

PHASE 6: TESTING & REFINEMENT (Week 12)

Goal: Make sure everything works perfectly

Task 6.1: User Testing

What to do:

- Have actual board members use the admin portal

- Watch where they get confused
- Fix issues
- Get feedback on features

Deliverable: List of bugs and improvements

Time: Ongoing

Task 6.2: Documentation

What to do: Create user manual for admin portal

Include:

- How to log in
- How to add/edit each type of content
- Screenshots for each major task
- Troubleshooting common issues
- Who to contact for help

Deliverable: User documentation

Time: 2-3 days

Task 6.3: Final Polish

What to do:

- Fix all known bugs

- Improve performance
- Add any missing features
- Final design tweaks

Deliverable: Production-ready system

Time: 1 week

TIMELINE SUMMARY

Total Duration: 12 weeks (3 months)

- Week 1: Planning & Setup
- Weeks 2-4: Backend Development (3 weeks)
- Weeks 5-8: Admin Portal (4 weeks)
- Weeks 9-10: Public Website (2 weeks)
- Week 11: Deployment (1 week)
- Week 12: Testing & Refinement (1 week)

This assumes:

- You work 15-20 hours per week
- You're learning as you go
- Some tasks may take longer as you're new to this

You can speed up by:

- Working more hours per week

- Getting help from friends
- Using more templates/libraries
- Simplifying features

CRITICAL SUCCESS FACTORS

1. **Don't skip planning** - Week 1 is crucial
2. **Test as you go** - Don't build everything then test
3. **One feature at a time** - Don't try to do everything simultaneously
4. **Ask for help** - When stuck for more than 2 hours, ask
5. **Commit to GitHub regularly** - Don't lose your work
6. **Keep it simple first** - Add fancy features later

WHEN YOU GET STUCK

Backend issues:

- Check Prisma documentation
- Check Express documentation
- Search error messages on Stack Overflow

Frontend issues:

- Check React documentation
- Check Material-UI documentation

- Look for similar examples on CodeSandbox

Database issues:

- Check pgAdmin to see actual data
- Look at Prisma Studio (visual database viewer)

Deployment issues:

- Check Railway/Vercel logs
- Verify environment variables
- Test API endpoints individually

NEXT IMMEDIATE STEPS

This week:

1. ✅ Setup complete (you did this!)
2. Draw database schema on paper
3. Share schema for review
4. Create remaining tables in Prisma
5. Build authentication endpoints

By end of Week 2:

- All database tables created
- All basic CRUD endpoints for Members working
- Tested in Postman

Then move to Week 3 tasks...

Remember: This is a marathon, not a sprint. Take it one task at a time. You've got this!