Yus Montessori School Management System

Complete Project Plan & Implementation Guide

🎯 Project Overview

Yus Montessori School Management System is a comprehensive AI-powered school administration platform designed to streamline operations, enhance parent-teacher communication, and automate administrative tasks for Yus Montessori School.

Core Objectives

- Reduce administrative burden by 70% through AI automation
- Centralize student, parent, and school communications
- Automate payment processing and financial tracking
- Enhance parent engagement through smart notifications
- Provide predictive analytics for enrollment and operations

🛠️ Technology Stack

Frontend Stack (Same as Pollify/CBG)

```json
{
  "framework": "Next.js 14 (App Router)",
  "runtime": "React 18",
  "language": "TypeScript",
  "styling": "Tailwind CSS",
  "ui_components": "Shadcn/ui + Radix UI",
  "state_management": "Zustand",
  "forms": "React Hook Form + Zod validation",
  "http_client": "Axios",
  "date_handling": "date-fns",
  "icons": "Lucide React",
  "charts": "Recharts",
  "animations": "Framer Motion",
  "notifications": "React Hot Toast"
}
```

Backend Stack (Same as CBG webapp)

```json
{
  "runtime": "Node.js 18+",
  "framework": "Express.js",
  "language": "JavaScript (ES6+)",
  "database": "MongoDB with Mongoose",
  "authentication": "JWT + bcrypt",
  "file_uploads": "Multer",
  "email": "Nodemailer + Gmail API",
  "payments": "Stripe",
  "ai_services": "OpenAI GPT-4",
  "pdf_generation": "PDFLib",
  "logging": "Winston",
  "validation": "Express-validator",
  "security": "Helmet + CORS + Rate Limiting"
}
```

Deployment & Infrastructure

```json
{
  "hosting": "Vercel (Frontend) + Railway/Render (Backend)",
  "database": "MongoDB Atlas",
  "file_storage": "Cloudinary or AWS S3",
  "domain": "Custom domain with SSL",
  "monitoring": "Sentry (Error tracking)",
  "analytics": "Plausible or Google Analytics"
}
```

📦 Project Modules

Module 1: Authentication & User Management

Status: ✅ Backend Complete | 🔄 Frontend Needed

Backend Components (✅ Complete)

- User model with role-based access (Admin/Teacher/Parent)

- JWT authentication with refresh tokens
- Password reset and email verification
- Account lockout and security features
- Profile management

Frontend Components (🔄 To Build)

```typescript
// Components to create:
- LoginForm.tsx
- RegisterForm.tsx
- ForgotPasswordForm.tsx
- ResetPasswordForm.tsx
- ProfileSettings.tsx
- UserDashboard.tsx
- RoleBasedNavigation.tsx
```

User Stories

- **Parent**: "I can securely log in and access my children's information"
- **Teacher**: "I can manage my class and communicate with parents"
- **Admin**: "I can manage all users and have full system access"

Module 2: Student Information System

Status: ✅ Backend Complete | 🔄 Frontend Needed

Backend Components (✅ Complete)

- Comprehensive student model with Montessori focus
- Parent-student relationships
- Health and emergency information
- Attendance tracking
- Communication logging

Frontend Components (🔄 To Build)

```typescript
```

```
// Components to create:
- StudentList.tsx
- StudentProfile.tsx
- StudentForm.tsx
- AttendanceTracker.tsx
- HealthInformation.tsx
- EmergencyContacts.tsx
- StudentSearch.tsx
- ClassRoster.tsx
- ObservationForm.tsx
- PortfolioViewer.tsx
```

User Stories

- **Parent**: "I can view my child's progress, observations, and attendance"
- **Teacher**: "I can record observations and track student development"
- **Admin**: "I can manage all student records and enrollment"

Module 3: AI-Powered Email Management

Status: ✅ Backend Complete | 🔄 Frontend Needed

Backend Components (✅ Complete)

- Gmail API integration
- AI email categorization and sentiment analysis
- Automated response generation
- Email threading and conversation tracking
- Response templates and customization

Frontend Components (🔄 To Build)

```typescript
```

```
// Components to create:
- EmailDashboard.tsx
- EmailList.tsx
- EmailViewer.tsx
- EmailComposer.tsx
- ResponseGenerator.tsx
- EmailFilters.tsx
- EmailStats.tsx
- TemplateManager.tsx
- ConversationThread.tsx
- EmailProcessingQueue.tsx
```

User Stories

- **Admin**: "AI categorizes emails and suggests responses automatically"
- **Teacher**: "I can quickly respond to parent inquiries with AI assistance"
- **System**: "Urgent emails are flagged and prioritized automatically"

Module 4: Payment Processing & Financial Management

Status: ✅ Backend Complete | 🔄 Routes & Frontend Needed

Backend Components

- ✅ Payment model with Stripe integration
- ✅ Recurring payment automation
- ✅ Receipt and tax document generation
- ✅ Payment reminder system
- 🔄 Payment routes (need to create)

Frontend Components (🔄 To Build)

```typescript
```

```
// Components to create:
- PaymentDashboard.tsx
- PaymentForm.tsx
- PaymentHistory.tsx
- InvoiceViewer.tsx
- ReceiptDownloader.tsx
- RecurringPaymentSetup.tsx
- PaymentReminders.tsx
- FinancialReports.tsx
- TaxDocuments.tsx
- RefundProcessor.tsx
```

User Stories

- **Parent**: "I can easily pay tuition online and download receipts"
- **Admin**: "Payments are processed automatically with proper tax documentation"
- **System**: "Late payments trigger automated reminders"

Module 5: Waitlist Management

Status: ✅ Backend Complete | 🔄 Routes & Frontend Needed

Backend Components

- ✅ Waitlist model with AI insights
- ✅ Position management and priority handling
- ✅ Conversion prediction algorithms
- 🔄 Waitlist routes (need to create)

Frontend Components (🔄 To Build)

typescript

```
// Components to create:
- WaitlistDashboard.tsx
- WaitlistForm.tsx
- WaitlistPosition.tsx
- ConversionPredictions.tsx
- TourScheduler.tsx
- WaitlistCommunications.tsx
- PositionManager.tsx
- WaitlistAnalytics.tsx
```

User Stories

- **Prospective Parent**: "I can join the waitlist and track my position"
- **Admin**: "AI helps predict which families are most likely to enroll"
- **System**: "Waitlist positions are managed automatically"

Module 6: Expense Tracking & Tax Management

Status: ✅ Backend Complete | 🔄 Routes & Frontend Needed

Backend Components

- ✅ Expense model with tax categorization
- ✅ Receipt management and storage
- ✅ Canadian tax compliance features
- 🔄 Expense routes (need to create)

Frontend Components (🔄 To Build)

```typescript
// Components to create:
- ExpenseTracker.tsx
- ExpenseForm.tsx
- ReceiptUploader.tsx
- TaxCategoryManager.tsx
- ExpenseReports.tsx
- TaxSummary.tsx
- VendorManager.tsx
- ApprovalWorkflow.tsx
```

User Stories

- **Admin**: "I can track all school expenses with proper tax categorization"
- **Accountant**: "Tax reports are generated automatically for year-end"
- **Staff**: "I can submit expense claims with receipt uploads"

Module 7: Newsletter & Communication System

Status: ✅ Backend Complete | 🔄 Routes & Frontend Needed

Backend Components

- ✅ Newsletter model with AI content generation
- ✅ Recipient management and segmentation
- ✅ Delivery tracking and analytics
- 🔄 Newsletter routes (need to create)

Frontend Components (🔄 To Build)

```typescript
// Components to create:
- NewsletterEditor.tsx
- ContentGenerator.tsx
- RecipientManager.tsx
- NewsletterPreview.tsx
- DeliveryScheduler.tsx
- NewsletterAnalytics.tsx
- TemplateLibrary.tsx
- CommunicationHub.tsx
```

User Stories

- **Admin**: "AI generates newsletter content from school activities"
- **Parents**: "I receive relevant school updates and announcements"
- **Teachers**: "I can contribute content for newsletters easily"

Module 8: Dashboard & Analytics

Status: 🔄 Complete Module Needed

Components to Build

```typescript
// Components to create:
- AdminDashboard.tsx
- TeacherDashboard.tsx
- ParentDashboard.tsx
- AnalyticsOverview.tsx
- EnrollmentTrends.tsx
- FinancialSummary.tsx
- EmailMetrics.tsx
- StudentProgress.tsx
- QuickActions.tsx
- NotificationCenter.tsx
```

User Stories

- **Admin**: "I see a complete overview of school operations at a glance"
- **Teacher**: "My dashboard shows my class information and tasks"
- **Parent**: "I have easy access to my children's information and school updates"

🗂️ Project Structure

```
yus-montessori-school/
├── 📁 frontend/                # Next.js App
│   ├── 📁 app/                 # App Router
│   │   ├── 📁 (auth)/          # Auth pages
│   │   ├── 📁 dashboard/       # Dashboard pages
│   │   ├── 📁 students/        # Student pages
│   │   ├── 📁 emails/          # Email pages
│   │   ├── 📁 payments/        # Payment pages
│   │   ├── 📁 waitlist/        # Waitlist pages
│   │   ├── 📁 expenses/        # Expense pages
│   │   └── 📁 newsletters/     # Newsletter pages
│   ├── 📁 components/          # Reusable components
│   │   ├── 📁 ui/              # Shadcn/ui components
│   │   ├── 📁 forms/           # Form components
│   │   ├── 📁 charts/          # Chart components
│   │   └── 📁 layout/          # Layout components
│   ├── 📁 lib/                 # Utilities
```

```
│   │   ├── api.ts              # API client
│   │   ├── auth.ts             # Auth helpers
│   │   ├── utils.ts            # Utility functions
│   │   └── validations.ts      # Zod schemas
│   ├── 📁 stores/              # Zustand stores
│   ├── 📁 hooks/               # Custom React hooks
│   └── 📁 types/               # TypeScript definitions
│
├── 📁 backend/                 # Express API
│   ├── server.js               # ✅ Entry point
│   ├── 📁 config/              # ✅ Configuration
│   ├── 📁 models/              # ✅ Database models
│   ├── 📁 routes/              # ✅ API routes (3/7 done)
│   ├── 📁 middleware/          # ✅ Middleware
│   ├── 📁 services/            # ✅ Business logic
│   ├── 📁 utils/               # ✅ Utilities
│   ├── 📁 uploads/             # File uploads
│   └── 📁 logs/                # Log files
│
├── 📁 docs/                    # Documentation
├── 📁 tests/                   # Test files
├── package.json                # Root dependencies
├── docker-compose.yml          # Docker setup
└── README.md                   # Project documentation
```

🚀 Implementation Phases

Phase 1: Foundation Setup (Week 1)

Goal: Complete backend and basic frontend setup

Day 1-2: Complete Backend

- ✅ Already done: 22 files created
- 🔄 Create remaining 4 route files
- 🔄 Add comprehensive API testing
- 🔄 Set up development environment

Day 3-5: Frontend Foundation

- Set up Next.js 14 with TypeScript

- Configure Tailwind CSS and Shadcn/ui
- Set up Zustand stores
- Create basic layout components
- Implement authentication flow

Day 6-7: Authentication & Basic UI

- Build login/register forms
- Create role-based navigation
- Set up protected routes
- Basic dashboard layouts

Phase 2: Core Modules (Week 2-3)

Goal: Implement primary functionality

Week 2: Student & Email Management

- Student information system
- Email dashboard with AI features
- Basic CRUD operations
- Search and filtering

Week 3: Payments & Communications

- Payment processing interface
- Email composer and response generator
- Communication tracking
- Basic reporting

Phase 3: Advanced Features (Week 4)

Goal: AI features and automation

AI Integration

- Email processing automation
- Newsletter content generation
- Waitlist predictions
- Smart notifications

Analytics & Reporting

- Dashboard widgets
- Financial reports
- Usage analytics
- Performance metrics

Phase 4: Polish & Deployment (Week 5)

Goal: Production-ready system

Testing & QA

- Unit tests for critical functions
- Integration testing
- User acceptance testing
- Performance optimization

Deployment

- Production environment setup
- Domain configuration
- SSL certificates
- Monitoring setup

🔧 Development Setup Instructions

Prerequisites

```bash
# Required software
- Node.js 18+
- MongoDB (local or Atlas)
- Git
- Code editor (VS Code recommended)
```

Backend Setup

```bash
```

```
# 1. Clone and setup backend
mkdir yus-montessori-school
cd yus-montessori-school
mkdir backend
cd backend

# 2. Copy all our created backend files
# (Use the 22 files we've already created)

# 3. Install dependencies
npm install

# 4. Set up environment variables
cp .env.example .env
# Fill in your actual values:
# - MongoDB connection string
# - JWT secret
# - Gmail API credentials
# - Stripe keys
# - OpenAI API key

# 5. Start development server
npm run dev
```

Frontend Setup

bash

```
# 1. Create Next.js app in project root
cd ..
npx create-next-app@latest frontend --typescript --tailwind --app

# 2. Install additional dependencies
cd frontend
npm install @radix-ui/react-* zustand react-hook-form zod axios date-fns lucide-react recharts framer-motion rea

# 3. Set up Shadcn/ui
npx shadcn-ui@latest init

# 4. Install common components
npx shadcn-ui@latest add button card input label select textarea dialog dropdown-menu

# 5. Start frontend development
npm run dev
```

Database Setup

```
bash

# Option 1: MongoDB Atlas (Recommended)
# 1. Create account at mongodb.com/atlas
# 2. Create new cluster
# 3. Get connection string
# 4. Add to .env file

# Option 2: Local MongoDB
# 1. Install MongoDB locally
# 2. Start MongoDB service
# 3. Use connection string: mongodb://localhost:27017/yus_montessori
```

📝 API Documentation

Authentication Endpoints

```
typescript
```

```
POST   /api/auth/register       # User registration
POST   /api/auth/login          # User login
POST   /api/auth/logout         # User logout
GET    /api/auth/me             # Get current user
PUT    /api/auth/profile        # Update profile
PUT    /api/auth/change-password # Change password
POST   /api/auth/forgot-password # Request password reset
POST   /api/auth/reset-password  # Reset password
```

Student Management Endpoints

typescript

```
GET    /api/students            # List students (paginated)
POST   /api/students            # Create student
GET    /api/students/:id        # Get student by ID
PUT    /api/students/:id        # Update student
DELETE /api/students/:id        # Archive student
POST   /api/students/:id/observations    # Add observation
POST   /api/students/:id/communications  # Add communication
GET    /api/students/:id/payments       # Get payment summary
GET    /api/students/:id/attendance     # Get attendance records
```

Email Management Endpoints

typescript

```
GET    /api/emails              # List emails (paginated)
POST   /api/emails/fetch        # Fetch new emails from Gmail
GET    /api/emails/:id          # Get email by ID
POST   /api/emails/:id/process   # Process email with AI
PATCH  /api/emails/:id/flags     # Update email flags
POST   /api/emails/:id/notes     # Add internal note
POST   /api/emails/:id/generate-response # Generate AI response
POST   /api/emails/:id/respond   # Send email response
PATCH  /api/emails/:id/link      # Link to student/parent
GET    /api/emails/stats/overview # Get email statistics
```

🎨 UI/UX Design Guidelines

Design Principles

- **Clean & Professional**: Suitable for educational environment
- **Accessible**: WCAG 2.1 compliant
- **Mobile-First**: Responsive design for all devices
- **Consistent**: Using Shadcn/ui design system
- **Efficient**: Minimize clicks for common tasks

Color Scheme

```css
/* Primary Colors (Montessori-inspired earth tones) */
--primary: 45 100% 20%      /* Deep forest green */
--primary-foreground: 0 0% 98%
--secondary: 35 25% 90%     /* Warm beige */
--secondary-foreground: 0 0% 20%

/* Accent Colors */
--accent: 25 95% 53%        /* Warm orange */
--muted: 35 10% 95%         /* Light neutral */
--border: 35 20% 85%        /* Soft border */
```

Typography

```css
/* Font Stack */
font-family: Inter, -apple-system, BlinkMacSystemFont, sans-serif;

/* Scale */
--text-xs: 0.75rem
--text-sm: 0.875rem
--text-base: 1rem
--text-lg: 1.125rem
--text-xl: 1.25rem
--text-2xl: 1.5rem
--text-3xl: 1.875rem
```

Component Examples

```typescript
// Primary Button
<Button className="bg-green-700 hover:bg-green-800">
  Save Student
</Button>

// Card Layout
<Card className="border-green-200">
  <CardHeader className="bg-green-50">
    <CardTitle>Student Information</CardTitle>
  </CardHeader>
  <CardContent className="p-6">
    {/* Content */}
  </CardContent>
</Card>

// Status Indicators
<Badge variant={status === 'active' ? 'default' : 'secondary'}>
  {status}
</Badge>
```

🧪 Testing Strategy

Testing Pyramid

Unit Tests (60%)

```javascript
// Test utilities and helper functions
// Test individual components in isolation
// Test service functions and API calls

// Example test files:
// - __tests__/utils/helpers.test.js
// - __tests__/components/StudentForm.test.tsx
// - __tests__/services/aiService.test.js
```

Integration Tests (30%)

```javascript
// Test API endpoints with database
// Test component interactions
// Test user workflows

// Example test files:
// - __tests__/api/students.test.js
// - __tests__/workflows/enrollment.test.tsx
```

E2E Tests (10%)

```javascript
// Test critical user journeys
// Test payment processing
// Test email workflows

// Tools: Playwright or Cypress
```

Test Setup

```bash
# Backend testing
npm install --save-dev jest supertest

# Frontend testing
npm install --save-dev @testing-library/react @testing-library/jest-dom

# E2E testing
npm install --save-dev @playwright/test
```

📊 Performance & Optimization

Frontend Optimization

- Next.js App Router for optimal loading
- Image optimization with next/image
- Code splitting and lazy loading
- Zustand for efficient state management

- React.memo for expensive components

Backend Optimization

- MongoDB indexing for frequent queries
- Redis caching for session and frequently accessed data
- Compression middleware for API responses
- Rate limiting to prevent abuse
- Connection pooling for database

Monitoring & Analytics

- Sentry for error tracking
- Winston logs with log rotation
- Performance monitoring with metrics
- User analytics with privacy compliance

🔒 Security Considerations

Authentication & Authorization

- JWT tokens with short expiration
- Role-based access control (RBAC)
- Password complexity requirements
- Account lockout after failed attempts
- Two-factor authentication (future enhancement)

Data Protection

- HTTPS everywhere (SSL/TLS)
- Input validation and sanitization
- XSS and CSRF protection
- Rate limiting and DDoS protection
- Regular security audits

Privacy Compliance

- PIPEDA compliance (Canadian privacy law)

- Data minimization principles
- Secure data deletion procedures
- Parent consent for student data
- Regular privacy impact assessments

📈 Scalability & Future Enhancements

Phase 2 Features (Future)

- Mobile app (React Native)
- Advanced reporting and analytics
- Calendar integration
- Parent communication portal
- Online learning modules
- Field trip management
- Photo sharing with parents
- Multi-school support

Technical Scalability

- Microservices architecture
- Docker containerization
- Kubernetes orchestration
- CDN for static assets
- Database sharding if needed
- Auto-scaling infrastructure

💰 Cost Estimation

Development Costs (5 weeks)

- Backend completion: 1 week
- Frontend development: 3 weeks
- Testing & deployment: 1 week
- **Total development time: ~200 hours**

Monthly Operating Costs

Hosting (Vercel + Railway):	$50-100/month
MongoDB Atlas:	$25-50/month
Stripe processing fees:	2.9% + $0.30/transaction
OpenAI API:	$20-100/month
Email service:	$10-30/month
Cloudinary/S3:	$10-25/month
Domain & SSL:	$15/month
Total estimated:	$130-320/month

✅ Success Metrics

User Adoption

- 90%+ parent login rate within first month
- 80% of emails processed automatically
- 95% payment success rate
- 50% reduction in administrative phone calls

Operational Efficiency

- 70% reduction in manual data entry
- 80% faster invoice processing
- 90% automated payment reminders
- 60% reduction in email response time

Financial Impact

- ROI positive within 6 months
- Reduced administrative staff hours
- Improved cash flow through automated billing
- Enhanced parent satisfaction scores

📞 Support & Maintenance

Launch Support (First Month)

- Daily monitoring and bug fixes
- User training sessions
- Data migration assistance
- 24/7 technical support

Ongoing Maintenance

- Monthly feature updates
- Security patches within 24 hours
- Performance monitoring and optimization
- User support through help desk
- Regular backups and disaster recovery testing

🎯 Getting Started Checklist

Immediate Actions (This Week)

- [ ] Complete remaining 4 backend route files
- [ ] Set up frontend Next.js project
- [ ] Configure development environment
- [ ] Create basic authentication flow
- [ ] Build first dashboard prototype

Week 2 Goals

- [ ] Complete student management interface
- [ ] Implement email processing dashboard
- [ ] Set up payment processing forms
- [ ] Create basic reporting features

Week 3 Goals

- [ ] AI feature integration
- [ ] Newsletter management system
- [ ] Waitlist and expense modules
- [ ] Mobile-responsive design

Week 4 Goals

- [ ] Testing and bug fixes
- [ ] Performance optimization
- [ ] Security audit
- [ ] Production deployment setup

Week 5 Goals

- [ ] Launch preparation
- [ ] User training materials
- [ ] Documentation completion
- [ ] Go-live support

This comprehensive plan provides a clear roadmap to build a production-ready school management system that will transform operations at Yus Montessori School while using proven technology stacks from your successful previous projects.