Student Portal - Functional Requirements

Overview

This document outlines the functional requirements for the **Student Portal** signup system, including reminder automation, admin dashboard, and compliance features.

Core Features: Student Portal

QR Code Landing Page

- Responsive design for mobile-first experience
- Seamless entry point for student registration

Training Class Selection

- Single class selection from **6 training types**
- User can only enroll in one class at a time

Input Validation

- Email **or** phone number required (choose one approach for implementation)
- Form validation on submission

Confirmation Flow

- Confirmation screen displaying signup details
- Allow users to review before final submission

Responsive Design

- **Top priority**: Mobile responsiveness
- Open question: Mobile only, or tablets too?

- *Note: Desktop and laptop screens are already handled*

Reminder System

Automated Scheduling

- Scheduling based on class-specific intervals
- Triggered by enrollment completion

Multi-Channel Delivery

- Email and SMS support
- Integration options: Twilio (Email), Amazon SES, SendGrid (SMS), Nodemailer
- *Start with Nodemailer for initial testing*

Message Templates

- Custom templates for each communication type
- Different templates for email vs. SMS
- Dynamic link injection (security considerations: per-student links, batch links, or universal links?)

Reliability

- Retry mechanisms for failed deliveries
- Fallback routing for critical messages

Admin Dashboard

Signup Management

- View all registered students in a paginated list
- Search functionality across student records
- *Note: Start with frontend pagination, migrate to backend as needed*

Filtering & Export

- Multi-criteria filtering for better UX
- CSV export functionality for student lists

Reminder Management (CRUD)

- Create, Read, Update, Delete operations for reminders
- *Remember: Use PATCH instead of PUT wherever possible*
- Template editor with live preview
- Opt-out management interface

Compliance & Privacy

Customer Controls

- One-click email unsubscribe links
- Opt-out database with enforcement

Regulatory Compliance

- GDPR compliance logging
- CAN-SPAM compliance logging
- Audit trails for all compliance-related actions

System Requirements

Delivery & Monitoring

- Delivery status tracking across all channels
- Error logging and monitoring
- Morgan/Winston logging in Dockerized setup

Audit & Security

- Audit trails for admin actions (CRUD and PATCH operations)
- Compliance and security support
- Tamper-proof logs for regulatory requirements

Health Checks

- Scheduled job health checks
- Automated alerts for job failures
- System uptime monitoring

Technical Architecture

Backend

Technology Stack

- **Runtime**: Node.js 18+
- **Framework**: Express.js
- **Database**: PostgreSQL
- **ORM**: Prisma
- **Job Queue**: Bull + Redis
- **Validation**: Zod
- **Authentication**: JWT + bcrypt
- **Email**: SendGrid
- **SMS**: Twilio
- **Logging**: Winston + Morgan

Architecture Pattern

Microservices with Three-Tier Architecture

Microservices architecture chosen for better fault tolerance over monolith.

Controller → Services → Repository Pattern

- **Repository Layer**: Handles all database operations
- **Services Layer**: Handles business logic (data modification, deletion, updating/patching)
- **Controllers Layer**: Handles client requests and forwards to services layer

This pattern ensures minimal LOC changes when modifications are needed.

Language Choice

JavaScript (not TypeScript) for initial implementation:

- Avoids unnecessary complexity
- Carefully typed JS > TS for this use case
- Faster compilation time
- TypeScript migration feasible at later stage

Database Design

```
-- students table
id (UUID, PK)
email (VARCHAR, indexed, nullable)
phone (VARCHAR, indexed, nullable)
opted_out_email (BOOLEAN, default: false)
opted_out_sms (BOOLEAN, default: false)
created_at (TIMESTAMP)
updated_at (TIMESTAMP)

-- signups table
id (UUID, PK)
student_id (UUID, FK -> students)
class_type (ENUM)
reminder_scheduled_date (TIMESTAMP)
reminder_sent_at (TIMESTAMP, nullable)
```

```
status (ENUM: pending, sent, failed)
created_at (TIMESTAMP)

-- message_templates table
id (UUID, PK)
class_type (ENUM)
channel (ENUM: email, sms)
subject (VARCHAR, nullable)
body (TEXT)
schedule_link (VARCHAR)
updated_at (TIMESTAMP)

-- delivery_logs table
id (UUID, PK)
signup_id (UUID, FK)
channel (ENUM)
status (ENUM: sent, failed, delivered, bounced)
provider_message_id (VARCHAR)
error_message (TEXT, nullable)
created_at (TIMESTAMP)

-- audit_logs table
id (UUID, PK)
admin_id (UUID)
action (VARCHAR)
resource_type (VARCHAR)
resource_id (UUID)
metadata (JSONB)
created_at (TIMESTAMP)
```

Note: Corrections and additions to be made on the fly

Background Job Architecture

- Queueing service + caching service for email and SMS reminders

- Options: Bull Queue, RabbitMQ, Redis PUB/SUB pattern
- Dead Letter Queue for failed jobs
- Rate limiting (custom logic or libraries like express-rate-limit)
- Separate queues for Emails/SMS/Failed Reminders to avoid conflicts

Security Considerations

- JWT authentication (custom implementation, not BetterAuth or Clerk)
  - Rationale: Economic and scalability reasons
- Custom error classes for consistent HTTP status codes
- Global error middleware (centralized)

Monitoring

- Prometheus for monitoring (post-MVP, finetuning stage)

Frontend

Technology Stack

- **Framework**: React 18+ (Vite for build)
- **Language**: JavaScript (TypeScript in the future)
- **Styling**: Chakra UI / Shadcn UI
- **Routing**: React Router
- **State Management**: Zustand
- **QR Code**: qrcode.react
- **Icons**: Lucide React

Project Structure

```
src/
  components/
    shared/
    student/
    admin/
  pages/
    StudentSignup.jsx
    AdminDashboard.jsx
    TemplateManager.jsx
    OptOutConfirmation.jsx
  store/ (global state manager)
    StudentStore.jsx
    AdminStore.jsx
    ReminderStore.jsx
  utils/
    formatters.jsx
    constants.jsx
```

Form Handling & Validation

- **React Hook Form**: Performance + validation
- **Zod Schema Validation**: Error-free database operations
- **Debounced Input Validation**: Real-time UX
- **Real-time Error Feedback**: Immediate user guidance

Mobile Optimizations

- Touch-friendly inputs
- Navbar shrinking into sidebar (hamburger menu) for better UX
- Progressive Web App (PWA) capabilities
- Viewport meta tag configuration

DevOps

Hosting & Containerization

- **Platform**: Amazon AWS
- **Containerization**: Docker
- **CI/CD**: GitHub Actions (for easier staging automation)

Open Questions

1. **Responsive Design**: Mobile-only or include tablet support?
2. **Contact Method**: Email or phone number implementation approach?
3. **Dynamic Links**: Per-student links, batch links, or universal links for scheduling?

Phase-Wise Development Plan

Phase 1: Project Setup

Backend Setup

1. Setup the Express Server with JavaScript
2. Configure the Prisma setup with PostgreSQL
3. Set up environment variables and dotenv configurations
4. Create the devised folder structure
5. Configure ESLint + Prettier setup
6. Configure Husky setup to ensure consistent coding style
7. Setup Dockerfiles for each microservice to ensure multi-stage builds

Frontend Setup

1. Vite + React project in JavaScript setup
2. Styling: TailwindCSS, Chakra/Shadcn setup

3. React Router setup
4. Axios setup with interceptors according to the modern setup

DevOps Setup

1. GitHub repo initialization
2. CI/CD pipeline skeleton
3. Environment variable management
4. Deploy staging environments

Phase 2: Database & Core Backend Setup

Database Schema

1. Prisma schema setup
2. Migration scripts
3. Seed data for development (mock data - please provide if available)
4. Database indexes optimization (as decided in the database tentative schema)

Authentication System

1. JWT authentication using tokens in cookies setup to mitigate security vulnerabilities
2. Admin login/register/refreshToken endpoints
3. Password hashing logic (likely bcryptJS)
4. Token refresh mechanism: Short-lived AccessToken, long-lived RefreshToken

Core API Development

1. API documentation (will attach soon once dev process starts, or might use SwaggerUI Docs)
2. Student validation logic + Zod schema validation + React Hook Forms
3. Error handling middleware
4. Request logging setup in the Controllers file for better tracking & observability

---

Phase 3: Student Portal Frontend

Landing Page

1. QR code generation/display
2. Mobile responsive hero section
3. Class selection UI component
4. Accessibility implementation

Registration Form

1. React Hook Form setup
2. Email/Phone validation
3. Progressive disclosure pattern
4. Real-time validation feedback
5. Error handling UI

Confirmation Screen

1. Success message component (through toast notifications)
2. Signup summary display (through modals)
3. Social share functionality (optional)

Integration Testing

1. Form submission flow
2. Error scenarios
3. Mobile responsiveness testing

Phase 4: Reminder Scheduling System

Job Queue Implementation

1. Bull Queue configuration
2. Redis connection setup
3. Job processor for reminders (Node-cron)

4. Retry logic implementation

Email Service

1. SendGrid integration (we already have the paid plan)
2. Template rendering engine
3. Unsubscribe link generation
4. Delivery tracking

SMS Service

1. Twilio API integration
2. Character limit handling
3. STOP command processing
4. Delivery status webhooks

Scheduler Logic

1. Cron job for checking due reminders
2. Batch processing optimizations
3. Rate limiting compliance

Phase 5: Admin Dashboard

Authentication & Layout

1. Protected route setup
2. Admin layout component
3. Navigation sidebar
4. Logout functionality

Signup Management

1. Data table with pagination
2. Search & filter implementation
3. Sorting functionality

4. Real-time status updates

Template Editor

1. Live preview functionality
2. Variable interpolation (dynamic value insertion)
3. Save/revert changes

CSV Export

1. Export service implementation
2. Column customization
3. Date range filtering
4. Download handler

Opt-Out Management

1. Opt-out and opt-in features
2. Preference management interface

Phase 6: Testing

1. Unit tests for core services
2. Integration tests for API endpoints
3. End-to-end testing for user flows
4. Performance testing
5. Security testing

Phase 7: Deployment

Infrastructure Setup

1. Caddy/Nginx for hosting
2. SSL certificate setup
3. Environment variable configuration

4. Database migration
5. Production environment setup

Optimization

1. Query optimization
2. Frontend bundle size reduction
3. CDN configuration (since our requirement is content heavy)

Monitoring & Maintenance

1. Health check endpoints
2. Error tracking setup
3. Performance monitoring
4. Backup and recovery procedures

SYSTEM DESIGN & WORKFLOW DIAGRAMS:

HIGH LEVEL SYSTEM ARCHITECTURE:

```
graph TB
    subgraph "Client Layer"
        A[Student Mobile/Web]
        B[Admin Dashboard]
        C[QR Code Scanner]
    end

    subgraph "API Gateway Layer"
        D[Express.js API Server]
        E[Rate Limiter]
        F[Auth Middleware]
    end

    subgraph "Application Layer"
        G[Signup Service]
        H[Reminder Service]
        I[Template Service]
        J[Opt-Out Service]
        K[Export Service]
    end

    subgraph "Background Jobs"
        L[Bull Queue Manager]
        M[Email Worker]
        N[SMS Worker]
        O[Scheduler Cron]
    end

    subgraph "Data Layer"
        P[(PostgreSQL)]
        Q[(Redis Cache)]
    end

    subgraph "External Services"
        R[SendGrid/AWS SES]
        S[Twilio SMS]
        T[Monitoring - Sentry]
    end

    A --> C
    C --> D
    B --> D
    D --> E
    E --> F
    F --> G
    F --> H
    F --> I
    F --> J
    F --> K

    G --> P
    H --> L
    I --> P
    J --> P
    K --> P

    L --> M
    L --> N
    L --> O

    M --> R
    N --> S

    G --> Q
    H --> Q

    D --> T
    M --> T
    N --> T
```

STUDENT SIGNUP WORKFLOW:

```
sequenceDiagram
    participant Student
    participant Frontend
    participant API
    participant Validator
    participant DB
    participant Queue

    Student->>Frontend: Scan QR Code / Visit Link
    Frontend->>Student: Display Class Selection Form
    Student->>Frontend: Select Class + Enter Contact Info
    Frontend->>Frontend: Client-side Validation

    alt Validation Fails
        Frontend->>Student: Show Error Messages
    else Validation Passes
        Frontend->>API: POST /api/signups
        API->>Validator: Validate Input Schema

        alt Invalid Data
            Validator->>API: Validation Error
            API->>Frontend: 400 Bad Request
            Frontend->>Student: Show Error
        else Valid Data
            Validator->>API: Data OK
            API->>DB: Check Existing Student (by email/phone)

            alt Student Exists
                DB->>API: Return Student ID
            else New Student
                DB->>API: Create Student Record
                DB->>API: Return New Student ID
            end

            API->>DB: Create Signup Record
            API->>DB: Calculate reminder_scheduled_date
            DB->>API: Signup Created

            API->>Queue: Schedule Reminder Job
            Queue->>API: Job ID

            API->>Frontend: 201 Created {signup_id, confirmat
```

```
ion}
            Frontend->>Student: Show Success Screen
        end
    end
```

Reminder Scheduling & Delivery Workflow:

```
flowchart TD
    A[Cron Job Runs Every Hour] --> B{Check DB for Due Remind
ers}
    B -->|No Reminders Due| C[Exit]
    B -->|Reminders Found| D[Fetch Batch of 100 Signups]

    D --> E{Check Opt-Out Status}
    E -->|User Opted Out| F[Mark as Skipped]
    E -->|User Active| G[Add to Processing Queue]

    G --> H{Contact Method?}
    H -->|Email Only| I[Queue Email Job]
    H -->|SMS Only| J[Queue SMS Job]
    H -->|Both| K[Queue Both Jobs]

    I --> L[Email Worker]
    J --> M[SMS Worker]
    K --> L
    K --> M

    L --> N[Render Email Template]
    N --> O[Inject Dynamic Variables]
    O --> P[SendGrid/SES API Call]
    P --> Q{Delivery Status}
    M --> R[Render SMS Template]
    R --> S[Character Limit Check]
    S --> T[Twilio API Call]
    T --> U{Delivery Status}

    Q -->|Success| V[Log Success + Update DB]
    Q -->|Failed| W[Retry Logic]
    W --> X{Retry Count < 3?}
    X -->|Yes| Y[Schedule Retry]
    X -->|No| Z[Log Permanent Failure]

    U -->|Success| V
    U -->|Failed| W

    V --> AA[Update signup.reminder_sent_at]
    Z --> AB[Send Alert to Admin]
```

ADMIN DASHBOARD FLOW:

```
flowchart LR
    A[Admin Login] --> B{JWT Valid?}
    B -->|No| C[Redirect to Login]
    B -->|Yes| D[Admin Dashboard]

    D --> E[View Signups Tab]
    D --> F[Manage Templates Tab]
    D --> G[Opt-Out Management Tab]
    D --> H[Export Tab]

    E --> E1[Apply Filters]
    E1 --> E2[Search by Email/Phone]
    E2 --> E3[Sort by Date/Class]
    E3 --> E4[Paginated Results]
    E4 --> E5{Actions}
    E5 -->|Edit| E6[Update Reminder Date]
    E5 -->|Delete| E7[Soft Delete Signup]
    E5 -->|Resend| E8[Manually Trigger Reminder]

    F --> F1[Select Class Type]
    F1 --> F2[Edit Email Template]
    F1 --> F3[Edit SMS Template]
    F2 --> F4[Live Preview]
    F3 --> F4
    F4 --> F5[Save Template]

    G --> G1[View Opted-Out Users]
    G1 --> G2{Action}
    G2 -->|Re-enable| G3[Update opt_out flag]
    G2 -->|Delete| G4[Remove from System]

    H --> H1[Select Date Range]
    H1 --> H2[Select Columns]
    H2 --> H3[Generate CSV]
    H3 --> H4[Download File]
```

OPT OUT HANDLING FLOW(UNSUBSRIBING):

```
sequenceDiagram
    participant User
    participant System
    participant DB
    participant Queue

    alt Email Unsubscribe
        User->>System: Click Unsubscribe Link
        System->>System: Verify Token
        System->>DB: UPDATE students SET opted_out_email=true
        DB->>System: Confirmation
        System->>User: Display Confirmation Page
    else SMS STOP Command
```

```
        User->>System: Reply "STOP" to SMS
        System->>System: Twilio Webhook Received
        System->>DB: Find Student by Phone Number
        DB->>System: Student Record
        System->>DB: UPDATE students SET opted_out_sms=true
        DB->>System: Confirmation
        System->>User: Send Confirmation SMS
    end

    System->>Queue: Cancel All Pending Jobs for User
    Queue->>System: Jobs Cancelled
    System->>DB: Log Opt-Out Event in audit_logs
```

DB SCHEMA DIAGRAM:

```
erDiagram
    STUDENTS ||--o{ SIGNUPS : registers
    SIGNUPS ||--o{ DELIVERY_LOGS : tracks
    MESSAGE_TEMPLATES ||--o{ SIGNUPS : uses
    ADMINS ||--o{ AUDIT_LOGS : creates

    STUDENTS {
        uuid id PK
        varchar email "nullable, indexed"
        varchar phone "nullable, indexed"
        boolean opted_out_email "default: false"
        boolean opted_out_sms "default: false"
        timestamp created_at
        timestamp updated_at
    }

    SIGNUPS {
        uuid id PK
        uuid student_id FK
        enum class_type "indexed"
        timestamp reminder_scheduled_date "indexed"
        timestamp reminder_sent_at "nullable"
        enum status "pending, sent, failed"
        text notes "nullable"
        timestamp created_at
        timestamp updated_at
    }

    MESSAGE_TEMPLATES {
        uuid id PK
        enum class_type "unique with channel"
        enum channel "email or sms"
        varchar subject "nullable, for email"
        text body
        varchar schedule_link
        jsonb variables "nullable"
        timestamp updated_at
    }

    DELIVERY_LOGS {
        uuid id PK
        uuid signup_id FK
        enum channel
        enum status "sent, failed, delivered, bounced"
        varchar provider_message_id "indexed"
        text error_message "nullable"
        jsonb metadata "nullable"
        timestamp created_at
    }

    ADMINS {
        uuid id PK
        varchar email "unique"
        varchar password_hash
        varchar name
        enum role "super_admin, admin, viewer"
        timestamp last_login "nullable"
        timestamp created_at
    }

    AUDIT_LOGS {
        uuid id PK
        uuid admin_id FK
        varchar action
        varchar resource_type
        uuid resource_id
        jsonb metadata
        varchar ip_address
        timestamp created_at
    }
```

BACKGROUND JOB PROCESSING ARCHITECTURE:

```
flowchart TB
    subgraph "Job Creation"
        A[API Server] -->|Add Job| B[Redis Queue]
        C[Cron Scheduler] -->|Add Job| B
    end

    subgraph "Job Queue (Bull)"
        B --> D{Job Type}
        D -->|Email| E[Email Queue]
        D -->|SMS| F[SMS Queue]
        D -->|Cleanup| G[Cleanup Queue]
    end

    subgraph "Worker Processes"
```

```
        E --> H[Email Worker 1]
        E --> I[Email Worker 2]
        F --> J[SMS Worker 1]
        F --> K[SMS Worker 2]
        G --> L[Cleanup Worker]
    end

    subgraph "Job Processing"
        H --> M{Check Opt-Out}
        I --> M
        J --> M
        K --> M

        M -->|Opted Out| N[Skip & Log]
        M -->|Active| O[Process Job]

        O --> P{Success?}
        P -->|Yes| Q[Complete Job]
        P -->|No| R{Retry Count < 3?}
        R -->|Yes| S[Retry with Backoff]
        R -->|No| T[Move to Dead Letter Queue]

        S --> B
    end

    subgraph "Monitoring"
        Q --> U[Update Metrics]
        T --> V[Alert Admin]
        N --> U
    end

    style B fill:#ffeb99,stroke:#333,stroke-width:2px
    style T fill:#ff9999,stroke:#333,stroke-width:2px
    style Q fill:#99ff99,stroke:#333,stroke-width:2px
```

API REQUEST FLOW(WITH CACHING):

```
sequenceDiagram
    participant Client
    participant LB as Load Balancer
    participant API as API Server
    participant Cache as Redis Cache
    participant DB as PostgreSQL
    participant Queue

    Client->>LB: GET /api/signups?page=1
    LB->>API: Route Request

    API->>API: Authenticate JWT
    API->>Cache: Check Cache Key "signups:page:1"

    alt Cache Hit
        Cache->>API: Return Cached Data
        API->>Client: 200 OK + Data
    else Cache Miss
        Cache->>API: null
        API->>DB: Query Signups (LIMIT, OFFSET)
        DB->>API: Return Data
        API->>Cache: Set Cache (TTL: 5 min)
        API->>Client: 200 OK + Data
    end

    Note over Client,Queue: Mutation Request (POST/PUT/DELET
E)

    Client->>LB: POST /api/signups
    LB->>API: Route Request
    API->>API: Validate Input
    API->>DB: INSERT INTO signups
    DB->>API: Created
    API->>Cache: Invalidate "signups:*" keys
    API->>Queue: Add Reminder Job
    Queue->>API: Job Scheduled
    API->>Client: 201 Created
```

SCALABILITY CONSIDERATIONS:

```
mindmap
  root((Scalability))
    Horizontal Scaling
      Load Balancer
        Round Robin
        Least Connections
        Health Checks
      Stateless API Servers
        JWT for Auth
        No Session Storage
      Auto-scaling Rules
        CPU > 70%
        Memory > 80%
        Request Rate
    Database Scaling
      Read Replicas
        Separate Read/Write
        Eventually Consistent Reads
      Connection Pooling
        pg pool max: 20
        Idle timeout: 30s
      Indexing Strategy
        email, phone indexed
        class_type indexed
        reminder_scheduled_date indexed
    Caching Strategy
      Redis Cache
```

```
        Signup Lists TTL 5min
        Template Cache TTL 1hr
      CDN for Static Assets
        Frontend Bundle
        Images
        Fonts
    Queue Scaling
      Multiple Workers
        Email workers: 3
        SMS workers: 3
      Rate Limiting
        Twilio: 100 req/s
        SendGrid: 10k/hr
```

RELIABILITY & FAULT TOLERANCE:

```
flowchart TD
    subgraph "Failure Scenarios"
        A[Database Connection Lost]
        B[External API Down]
        C[Worker Process Crash]
        D[Network Timeout]
    end

    subgraph "Mitigation Strategies"
        A --> E[Connection Retry with Exponential Backoff]
        E --> F[Fallback to Read Replica]
        F --> G[Circuit Breaker Pattern]

        B --> H[Queue Job for Retry]
        H --> I[Alternative Provider]
        I --> J[Dead Letter Queue after 3 attempts]

        C --> K[PM2 Auto-restart]
        K --> L[Health Check Endpoint]
        L --> M[Alert on Repeated Failures]
        D --> N[Request Timeout: 30s]
        N --> O[Retry with Backoff]
        O --> P[Fail Gracefully with User Message]
    end

    subgraph "Monitoring"
        Q[Uptime Checks Every 1min]
        R[Error Rate Alerts > 5%]
        S[Job Queue Depth Alert > 1000]
        T[Database Replication Lag]
    end

    style J fill:#ff9999,stroke:#333,stroke-width:2px
    style M fill:#ff9999,stroke:#333,stroke-width:2px
```

SECURITY ARCHITECTURE:

```
graph TB
    subgraph "Request Security Layers"
        A[Client Request] --> B[HTTPS/TLS 1.3]
        B --> C[Rate Limiting Middleware]
        C --> D[CORS Validation]
        D --> E[Helmet.js Security Headers]
        E --> F[Input Sanitization]
        F --> G[Schema Validation]
    end

    subgraph "Authentication Layer"
        G --> H{Authenticated Route?}
        H -->|Yes| I[JWT Verification]
        H -->|No| J[Public Route Handler]
        I --> K{Token Valid?}
        K -->|No| L[401 Unauthorized]
        K -->|Yes| M[Extract User Claims]
        M --> N{Admin Role?}
        N -->|No| O[403 Forbidden]
        N -->|Yes| P[Route Handler]
    end

    subgraph "Data Protection"
        P --> Q[Parameterized Queries]
        Q --> R[PII Encryption at Rest]
        R --> S[Audit Logging]
    end

    subgraph "Opt-Out Token Security"
        T[Generate Unsubscribe Link] --> U[HMAC-SHA256 Signat
ure]
        U --> V[Time-limited Token 30 days]
        V --> W[Single-use Token]
    end

    style L fill:#ff9999,stroke:#333,stroke-width:2px
    style O fill:#ff9999,stroke:#333,stroke-width:2px
```