Technical Specification

System Requirements

Performance Requirements

- **Concurrent Users**: Support 50+ active rooms simultaneously
- **Response Time**: < 200ms for API responses, < 50ms for WebSocket events
- **Availability**: 99.9% uptime during school hours
- **Real-time Latency**: < 100ms for submission approvals and roster updates

Scalability Requirements

- **Room Capacity**: 50 students per room maximum
- **Session Duration**: Support 8+ hour continuous sessions
- **Data Retention**: 1 year of participation history
- **Export Capability**: Handle CSV exports up to 10,000 records

Frontend Technical Specification

Technology Stack

- **Framework**: Next.js 14.2.28 with App Router
- **Language**: TypeScript 5.2.2
- **Styling**: Tailwind CSS 3.3.3
- **UI Components**: Radix UI primitives with shadcn/ui
- **State Management**: React hooks with Context API
- **Real-time**: Socket.io-client 4.7.2

Component Architecture

Core Pages

Student Landing (`/student`)

```
interface StudentLandingProps {
  onRoomJoin: (roomCode: string) => void
  validationError?: string
  isLoading: boolean
}

// Key features:
// - Room code input with validation
// - Columned student list with radio button selection
// - Point selection dropdown at top of interface
// - Real-time room status checking
// - Error handling and user feedback
// - Mobile-optimized input interface
```

Teacher Dashboard (`/teacher`)

```
interface TeacherDashboardProps {
  rooms: Room[]
  onRoomCreate: (roomData: RoomCreationData) => void
  onRoomToggle: (roomId: string, isActive: boolean) => void
  onCSVUpload: (file: File) => Promise<Student[]>
}

// Key features:
// - Room creation with one-column CSV student roster upload
// - Active room management and monitoring
// - Quick access to presentation mode
// - Room settings and configuration
```

Presentation View (`/teacher/[roomCode]/presentation`)

```
interface PresentationViewProps {
  room: Room
  students: Student[]
  pendingSubmissions: Submission[]
  onApproval: (submissionId: string, approved: boolean) => void
  onReset: (type: ResetType, targetId?: string) => void
}

// Layout structure:
// - 85% student roster with real-time points
// - 15% approval queue (fixed position, compact design)
// - Responsive breakpoints for different screen sizes
// - Keyboard shortcuts for quick approvals
```

Shared Components

Real-time Provider

```
interface SocketContextType {
  socket: Socket | null
  isConnected: boolean
  joinRoom: (roomCode: string, role: 'teacher' | 'student') => void
  leaveRoom: () => void
  emit: (event: string, data: any) => void
}

const SocketProvider: React.FC<{ children: ReactNode }>
```

Approval Queue Component

```
interface ApprovalQueueProps {
  submissions: Submission[]
  onApprove: (id: string) => void
  onReject: (id: string) => void
  maxHeight?: string
  compact?: boolean
}

// Features:
// - Auto-scroll to new submissions
// - Keyboard navigation (Enter=approve, Escape=reject)
// - Bulk selection and action capabilities
// - Loading states and error handling
```

State Management

Room State

```
interface RoomState {
  roomCode: string | null
  isActive: boolean
  students: Student[]
  pendingSubmissions: Submission[]
  userRole: 'teacher' | 'student' | null
  connectionStatus: 'connected' | 'disconnected' | 'connecting'
}

const useRoomState = () => {
  const [state, dispatch] = useReducer(roomReducer, initialState)
  // WebSocket event handlers and state updates
}
```

Student Management

```
interface StudentState {
  id: string
  name: string
  totalPoints: number
  isOnline: boolean
  lastSubmissionAt?: Date
  pendingPoints?: number
}

const useStudentState = (studentId: string) => {
  // Individual student state management
  // Real-time point updates
  // Submission status tracking
}
```

Responsive Design Breakpoints

```
/* Mobile First Approach */
.responsive-layout {
  /* Mobile: < 640px */
  @apply flex-col space-y-4;

  /* Tablet: 640px - 1024px */
  @screen sm {
    @apply flex-row space-x-4 space-y-0;
  }

  /* Desktop: > 1024px */
  @screen lg {
    @apply grid grid-cols-3 gap-6;
  }
}

/* Presentation View Layouts */
.presentation-layout {
  /* Small screens: Tabbed interface */
  @apply block md:hidden;

  /* Medium screens: Stacked layout */
  @apply hidden md:block lg:hidden;

  /* Large screens: Side-by-side layout */
  @apply hidden lg:flex lg:space-x-6;
}
```

Backend Technical Specification

Technology Stack

- **Runtime**: Node.js 18+ with Express.js 4.18+
- **Language**: TypeScript with strict mode
- **Database**: PostgreSQL 14+ with Prisma ORM 6.7.0
- **Real-time**: Socket.io 4.7.2
- **Validation**: Zod schemas for type-safe validation
- **Authentication**: Room-based access control

API Specification

Room Management Endpoints

POST /api/rooms/create

```
// Request (FormData for CSV upload)
interface CreateRoomRequest {
  name: string
  csvFile: File // One-column CSV with student names
}

// Response
interface CreateRoomResponse {
  roomCode: string
  roomId: string
  studentsCreated: number
}

// Validation
const createRoomSchema = z.object({
  name: z.string().min(1).max(100),
  csvFile: z.instanceof(File).refine(
    file => file.type === 'text/csv' || file.name.endsWith('.csv'),
    'File must be a CSV'
  )
})
```

POST /api/rooms/validate

```
// Request
interface ValidateRoomRequest {
  roomCode: string
}

// Response
interface ValidateRoomResponse {
  isValid: boolean
  isActive: boolean
  roomName?: string
  error?: string
}

// Business Logic
const validateRoom = async (roomCode: string) => {
  // 1. Check room exists
  // 2. Verify room is active
  // 3. Return room metadata
  // 4. Log validation attempt
}
```

Submission Management Endpoints

POST /api/students/submit

```
// Request
interface SubmitPointsRequest {
  roomCode: string
  studentId: string
  points: 1 | 2 | 3
}

// Response
interface SubmitPointsResponse {
  submissionId: string
  status: 'pending' | 'error'
  queuePosition?: number
}

// Rate Limiting: 3 submissions per minute per student
```

POST /api/rooms/[roomCode]/approve

```
// Request
interface ApprovalRequest {
  submissionId: string
  approved: boolean
  bulkIds?: string[] // For bulk operations
}

// Response
interface ApprovalResponse {
  processedCount: number
  updatedStudents: Array<{
    studentId: string
    newTotal: number
  }>
}
```

Database Schema Implementation

Prisma Schema Definition

```
generator client {
  provider = "prisma-client-js"
}

datasource db {
  provider = "postgresql"
  url      = env("DATABASE_URL")
}

model Room {
  id              String   @id @default(cuid())
  roomCode        String   @unique
  name            String
  isActive        Boolean  @default(true)
  createdAt       DateTime @default(now())
  updatedAt       DateTime @updatedAt
  lastActivityAt  DateTime @default(now())

  students        Student[]
  submissions     Submission[]
  resetHistory    ResetHistory[]

  @@map("rooms")
}

model Student {
  id          String   @id @default(cuid())
  name        String
  roomId      String
  totalPoints Int      @default(0)
  isOnline    Boolean  @default(false)
  lastActive  DateTime @default(now())

  room        Room     @relation(fields: [roomId], references: [id], onDelete:
Cascade)
  submissions Submission[]

  @@unique([roomId, name])
  @@map("students")
}

model Submission {
  id          String           @id @default(cuid())
  studentId   String
  roomId      String
  points      Int              // 1, 2, or 3
  status      SubmissionStatus @default(PENDING)
  submittedAt DateTime         @default(now())
  processedAt DateTime?

  student     Student @relation(fields: [studentId], references: [id], onDelete: Cas-
cade)
  room        Room    @relation(fields: [roomId], references: [id], onDelete: Cascade)

  @@map("submissions")
}

model ResetHistory {
  id        String    @id @default(cuid())
  roomId    String
  type      ResetType
  targetId  String?   // studentId for individual resets
```

```
  timestamp DateTime  @default(now())
  metadata  Json?     // Store additional reset context

  room      Room @relation(fields: [roomId], references: [id], onDelete: Cascade)

  @@map("reset_history")
}

enum SubmissionStatus {
  PENDING
  APPROVED
  REJECTED
}

enum ResetType {
  STUDENT
  CLASS
  SESSION
}
```

Database Queries Optimization

High-Frequency Queries

```
-- Room validation (most frequent)
CREATE INDEX CONCURRENTLY idx_room_code_active
ON rooms (room_code, is_active);

-- Pending submissions by room
CREATE INDEX CONCURRENTLY idx_submission_pending
ON submissions (room_id, status, submitted_at)
WHERE status = 'PENDING';

-- Student roster with points
CREATE INDEX CONCURRENTLY idx_student_room_points
ON students (room_id, total_points DESC, name);

-- Recent activity tracking
CREATE INDEX CONCURRENTLY idx_room_activity
ON rooms (last_activity_at DESC)
WHERE is_active = true;
```

Optimized Query Examples

```
// Get room with pending submissions and student roster
const getRoomData = async (roomCode: string) => {
  return await prisma.room.findUnique({
    where: { roomCode },
    include: {
      students: {
        orderBy: [
          { totalPoints: 'desc' },
          { name: 'asc' }
        ]
      },
      submissions: {
        where: { status: 'PENDING' },
        include: { student: true },
        orderBy: { submittedAt: 'asc' }
      }
    }
  })
}

// Batch approval processing
const processBulkApprovals = async (
  submissionIds: string[],
  approved: boolean
) => {
  return await prisma.$transaction(async (tx) => {
    // Update submissions
    await tx.submission.updateMany({
      where: { id: { in: submissionIds } },
      data: {
        status: approved ? 'APPROVED' : 'REJECTED',
        processedAt: new Date()
      }
    })

    // Update student points if approved
    if (approved) {
      const submissions = await tx.submission.findMany({
        where: { id: { in: submissionIds } },
        include: { student: true }
      })

      for (const submission of submissions) {
        await tx.student.update({
          where: { id: submission.studentId },
          data: {
            totalPoints: {
              increment: submission.points
            }
          }
        })
      }
    }
  })
}
```

WebSocket Event Specification

Connection Management

```
interface SocketConnection {
  id: string
  roomCode: string
  role: 'teacher' | 'student'
  userId?: string
  connectedAt: Date
}

class SocketManager {
  private connections = new Map<string, SocketConnection>()
  private roomChannels = new Map<string, Set<string>>()

  handleConnection(socket: Socket) {
    socket.on('room:join', this.handleRoomJoin.bind(this))
    socket.on('room:leave', this.handleRoomLeave.bind(this))
    socket.on('submission:create', this.handleSubmission.bind(this))
    socket.on('approval:process', this.handleApproval.bind(this))
    socket.on('disconnect', this.handleDisconnect.bind(this))
  }
}
```

Event Definitions

Client → Server Events

```
// Room management
interface RoomJoinEvent {
  roomCode: string
  role: 'teacher' | 'student'
  studentId?: string
}

interface RoomLeaveEvent {
  roomCode: string
}

// Student submissions
interface SubmissionCreateEvent {
  roomCode: string
  studentId: string
  points: 1 | 2 | 3
}

// Teacher approvals
interface ApprovalProcessEvent {
  submissionIds: string[]
  approved: boolean
  roomCode: string
}
```

Server → Client Events

```typescript
// Real-time updates
interface RosterUpdateEvent {
  students: Array<{
    id: string
    name: string
    totalPoints: number
    isOnline: boolean
  }>
  timestamp: number
}

interface QueueUpdateEvent {
  submissions: Array<{
    id: string
    studentName: string
    points: number
    submittedAt: string
  }>
}

interface PointsUpdateEvent {
  studentId: string
  newTotal: number
  pointsAdded: number
}

// System events
interface ErrorEvent {
  message: string
  code: string
  context?: any
}

interface StatusEvent {
  type: 'room_activated' | 'room_deactivated' | 'connection_restored'
  data?: any
}
```

Security Implementation

Input Validation

```
// Zod schemas for all API inputs
export const schemas = {
  roomCode: z.string().regex(/^[A-Z0-9]{6}$/, 'Invalid room code format'),
  studentName: z.string().min(1).max(50).trim(),
  points: z.number().int().min(1).max(3),
  roomName: z.string().min(1).max(100).trim(),
}

// Validation middleware
export const validateInput = (schema: z.ZodSchema) => {
  return (req: Request, res: Response, next: NextFunction) => {
    try {
      req.body = schema.parse(req.body)
      next()
    } catch (error) {
      res.status(400).json({
        error: 'Validation failed',
        details: error.errors
      })
    }
  }
}
```

Rate Limiting

```
import { RateLimiter } from 'limiter'

// Different limits for different operations
const limits = {
  submission: new RateLimiter(3, 'minute'), // 3 submissions per minute
  roomCreation: new RateLimiter(5, 'hour'), // 5 rooms per hour
  validation: new RateLimiter(20, 'minute'), // 20 validations per minute
}

const applyRateLimit = (type: keyof typeof limits) => {
  return async (req: Request, res: Response, next: NextFunction) => {
    const identifier = req.ip + (req.body.studentId || req.body.roomCode || '')

    if (!limits[type].tryRemoveTokens(1)) {
      return res.status(429).json({
        error: 'Rate limit exceeded',
        retryAfter: limits[type].getTokensRemaining()
      })
    }

    next()
  }
}
```

Data Sanitization

```
// SQL injection prevention (handled by Prisma)
// XSS prevention
import DOMPurify from 'dompurify'

const sanitizeInput = (input: string): string => {
  return DOMPurify.sanitize(input.trim())
}

// Room code generation with security
const generateRoomCode = (): string => {
  const chars = 'ABCDEFGHJKLMNPQRSTUVWXYZ23456789' // Exclude confusing chars
  let result = ''
  for (let i = 0; i < 6; i++) {
    result += chars.charAt(Math.floor(Math.random() * chars.length))
  }
  return result
}
```

Performance Optimization

Database Connection Pooling

```
// Prisma connection configuration
const prisma = new PrismaClient({
  datasources: {
    db: {
      url: process.env.DATABASE_URL
    }
  },
  log: ['error', 'warn'],
  errorFormat: 'pretty'
})

// Connection pool settings in DATABASE_URL
// postgresql://user:password@host:5432/db?connection_limit=20&pool_timeout=20
```

Caching Strategy

```
// Redis configuration for session caching
import Redis from 'ioredis'

const redis = new Redis({
  host: process.env.REDIS_HOST || 'localhost',
  port: parseInt(process.env.REDIS_PORT || '6379'),
  retryDelayOnFailover: 100,
  maxRetriesPerRequest: 3
})

// Cache frequently accessed data
const cacheRoomData = async (roomCode: string, data: any) => {
  await redis.setex(`room:${roomCode}`, 300, JSON.stringify(data)) // 5min TTL
}

const getCachedRoomData = async (roomCode: string) => {
  const cached = await redis.get(`room:${roomCode}`)
  return cached ? JSON.parse(cached) : null
}
```

WebSocket Optimization

```
// Connection limits and cleanup
const SOCKET_CONFIG = {
  maxConnections: 1000,
  heartbeatInterval: 30000,
  heartbeatTimeout: 60000,
  cleanupInterval: 300000 // 5 minutes
}

// Efficient room broadcasting
const broadcastToRoom = (roomCode: string, event: string, data: any) => {
  const roomSockets = roomChannels.get(roomCode)
  if (roomSockets) {
    roomSockets.forEach(socketId => {
      const socket = connections.get(socketId)
      if (socket?.connected) {
        socket.emit(event, data)
      }
    })
  }
}
```

Error Handling and Logging

Error Categories

```
enum ErrorCodes {
  // Validation errors (400)
  INVALID_ROOM_CODE = 'INVALID_ROOM_CODE',
  INVALID_STUDENT_DATA = 'INVALID_STUDENT_DATA',
  RATE_LIMIT_EXCEEDED = 'RATE_LIMIT_EXCEEDED',

  // Authorization errors (403)
  ROOM_INACTIVE = 'ROOM_INACTIVE',
  UNAUTHORIZED_ACTION = 'UNAUTHORIZED_ACTION',

  // Not found errors (404)
  ROOM_NOT_FOUND = 'ROOM_NOT_FOUND',
  STUDENT_NOT_FOUND = 'STUDENT_NOT_FOUND',

  // Server errors (500)
  DATABASE_ERROR = 'DATABASE_ERROR',
  WEBSOCKET_ERROR = 'WEBSOCKET_ERROR',
  INTERNAL_ERROR = 'INTERNAL_ERROR'
}

class AppError extends Error {
  constructor(
    public code: ErrorCodes,
    public message: string,
    public statusCode: number = 500,
    public context?: any
  ) {
    super(message)
    this.name = 'AppError'
  }
}
```

Structured Logging

```
import winston from 'winston'

const logger = winston.createLogger({
  level: process.env.LOG_LEVEL || 'info',
  format: winston.format.combine(
    winston.format.timestamp(),
    winston.format.errors({ stack: true }),
    winston.format.json()
  ),
  transports: [
    new winston.transports.File({ filename: 'logs/error.log', level: 'error' }),
    new winston.transports.File({ filename: 'logs/combined.log' }),
    new winston.transports.Console({
      format: winston.format.simple()
    })
  ]
})

// Usage examples
logger.info('Room created', {
  roomCode: 'ABC123',
  studentCount: 25,
  userId: 'teacher_123'
})

logger.error('Database connection failed', {
  error: error.message,
  stack: error.stack,
  operation: 'room_creation'
})
```

This technical specification provides comprehensive implementation details for building a robust, scalable classroom participation tracking system that meets all requirements specified in the PRD.