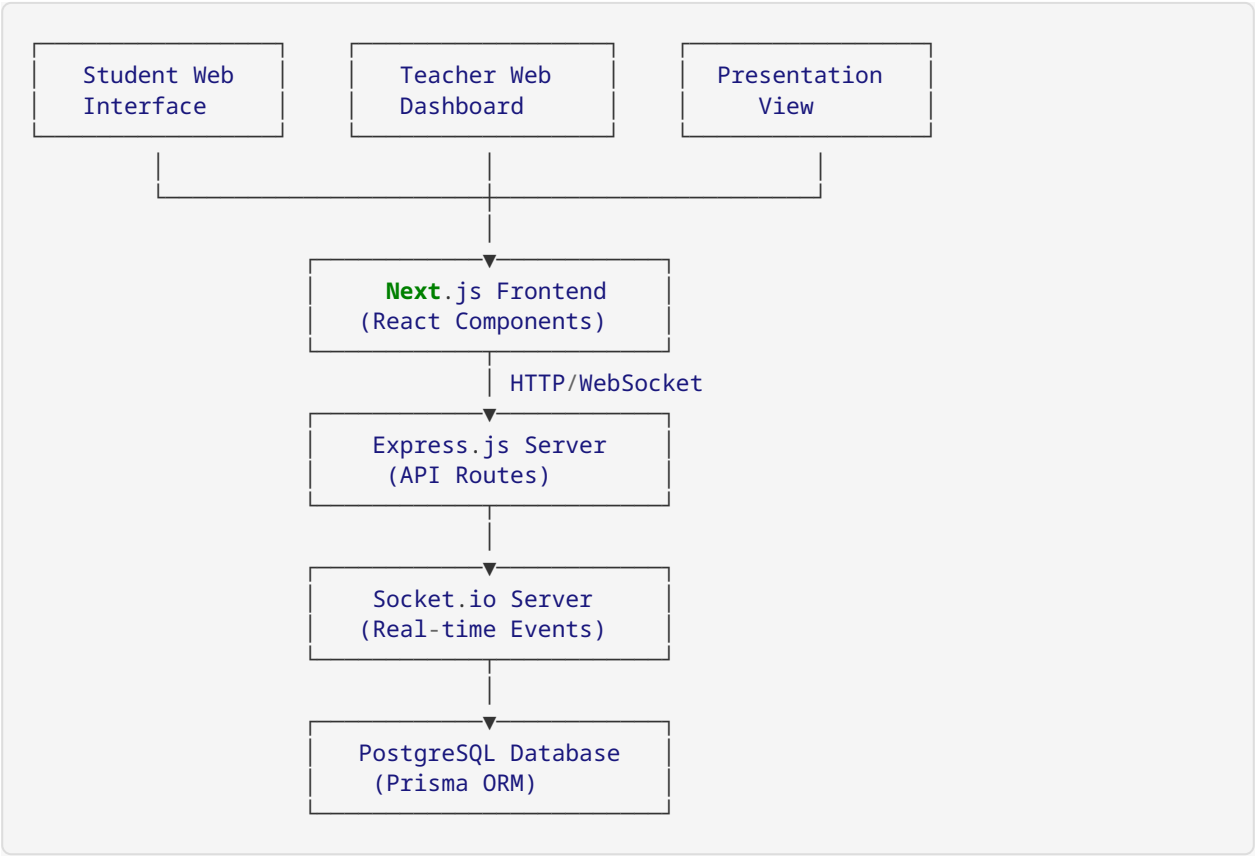
System Architecture

Overview

The Classroom Participation Tracker is built as a modern web application with real-time capabilities, designed to scale across multiple concurrent teacher sessions while maintaining simplicity and performance.

System Architecture Diagram



Component Architecture

Frontend Layer

Core Components

1. Student Interface (`/student`)
- `StudentLanding` : Room code entry and validation
- `StudentParticipation` : Point submission interface
- `StudentStatus` : Real-time feedback and current status

2. Teacher Dashboard (`/teacher`)
- `TeacherDashboard` : Room management and overview
- `RoomCreation` : New room setup and configuration
- `RoomManagement` : Active session controls and settings

3. Presentation View (`/teacher/[roomCode]/presentation`)
- `PresentationLayout` : Dual-panel layout management
- `StudentRoster` : Real-time class roster with points
- `ApprovalQueue` : Fixed-position approval interface
- `ResetControls` : Class and individual reset functionality

Shared Components

UI Components
- `Button` , `Input` , `Dialog` : Base UI elements
- `LoadingSpinner` : Async operation feedback
- `Toast` : User notification system
- `ConfirmDialog` : Safety confirmation modals

Real-time Components
- `SocketProvider` : WebSocket connection management
- `RealtimeUpdates` : Live data synchronization
- `ConnectionStatus` : Network status indicator

Backend Layer

API Routes Structure

```
/api/
├── rooms/
│   ├── create              # POST: Create new room
│   ├── validate            # POST: Validate room code
│   ├── [roomCode]/
│   │   ├── activate        # POST: Toggle room status
│   │   ├── students        # GET: Retrieve room roster
│   │   ├── submissions     # GET: Pending approvals
│   │   ├── approve         # POST: Approve submission
│   │   ├── reject          # POST: Reject submission
│   │   └── reset           # POST: Reset operations
├── students/
│   ├── join                # POST: Join room session
│   ├── submit              # POST: Submit points
│   └── status              # GET: Current student status
└── export/
    └── csv                 # GET: Export room data
```

WebSocket Events

Client → Server Events

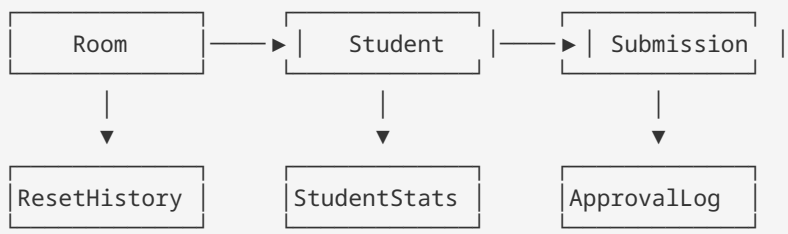
```
// Room Management
'room:join' -> { roomCode: string, studentId?: string }
'room:leave' -> { roomCode: string }

// Submissions
'submission:create' -> { studentId: string, points: number, roomCode: string }

// Teacher Actions
'approval:approve' -> { submissionId: string }
'approval:reject' -> { submissionId: string }
'room:reset' -> { roomCode: string, type: 'student' | 'class' | 'session' }
```

Server → Client Events

```
// Real-time Updates
'room:status' -> { isActive: boolean, participantCount: number }
'roster:update' -> { students: Student[], timestamp: number }
'queue:update' -> { submissions: Submission[] }
'points:update' -> { studentId: string, newTotal: number }

// System Events
'error:room' -> { message: string, code: string }
'connection:status' -> { status: 'connected' | 'disconnected' }
```

Database Architecture

Entity Relationship Diagram



Schema Details

Room Table

```
CREATE TABLE Room (
  id              String    PRIMARY KEY
  roomCode        String    UNIQUE NOT NULL
  name            String    NOT NULL
  isActive        Boolean   DEFAULT true
  createdAt       DateTime  DEFAULT now()
  updatedAt       DateTime  DEFAULT now()
  lastActivityAt  DateTime  DEFAULT now()
  students        Student[]
  submissions     Submission[]
  resetHistory    ResetHistory[]
)
```

Student Table

```
CREATE TABLE Student (
  id          String    PRIMARY KEY
  name        String    NOT NULL
  roomId      String    FOREIGN KEY → Room.id
  totalPoints Int       DEFAULT 0
  isOnline    Boolean   DEFAULT false
  lastActive  DateTime  DEFAULT now()
  submissions Submission[]
)
```

Submission Table

```
CREATE TABLE Submission (
  id          String           PRIMARY KEY
  studentId   String           FOREIGN KEY → Student.id
  roomId      String           FOREIGN KEY → Room.id
  points      Int              NOT NULL (1-3)
  status      SubmissionStatus DEFAULT 'PENDING'
  submittedAt DateTime         DEFAULT now()
  approvedAt  DateTime?
  rejectedAt  DateTime?
)

ENUM SubmissionStatus {
  PENDING
  APPROVED
  REJECTED
}
```

Database Optimizations

Indexing Strategy

```
-- High-frequency lookups
CREATE INDEX idx_room_code ON Room(roomCode);
CREATE INDEX idx_student_room ON Student(roomId);
CREATE INDEX idx_submission_status ON Submission(status, submittedAt);

-- Real-time queries
CREATE INDEX idx_active_rooms ON Room(isActive, lastActivityAt);
CREATE INDEX idx_pending_submissions ON Submission(status, roomId)
  WHERE status = 'PENDING';
```

Query Optimization

- Room validation: Single query with room code index
- Roster updates: Batch student queries with room filter
- Approval queue: Filtered pending submissions with time ordering
- Statistics: Aggregated queries with proper indexing

Real-Time Communication

WebSocket Connection Management

```
// Connection lifecycle
class SocketManager {
  // Teacher connections: room management and approvals
  teacherConnections: Map<string, Socket> = new Map()

  // Student connections: submissions and status updates
  studentConnections: Map<string, Socket> = new Map()

  // Room-specific channels for isolated updates
  roomChannels: Map<string, Set<Socket>> = new Map()
}
```

Event Broadcasting Strategy

Room-Scoped Events
- All clients in a room receive roster updates
- Only teacher connections receive approval queue updates
- Student-specific events sent to individual connections

Performance Considerations
- Maximum 100 connections per room (30 students + 70 observers)
- Heartbeat mechanism every 30 seconds
- Automatic reconnection with exponential backoff
- Connection pooling and cleanup on disconnect

Security Architecture

Access Control Model

Room-Based Security
1. **Public Access**: Landing pages and documentation
2. **Room Access**: Valid room code required for entry
3. **Teacher Access**: Room creation and management
4. **Student Access**: Submission and status viewing only

Data Validation Pipeline

```
// Input validation flow
Request → Rate Limiting → Schema Validation → Business Logic → Database
```

Validation Layers
- Rate limiting: 10 requests/minute per IP for submissions
- Schema validation: Zod schemas for all API inputs
- Business logic: Room status, student enrollment checks
- Database constraints: Foreign keys, unique constraints

Security Headers

```
// Next.js security configuration
const securityHeaders = [
  { key: 'X-DNS-Prefetch-Control', value: 'on' },
  { key: 'X-XSS-Protection', value: '1; mode=block' },
  { key: 'X-Frame-Options', value: 'SAMEORIGIN' },
  { key: 'X-Content-Type-Options', value: 'nosniff' },
  { key: 'Content-Security-Policy', value: cspHeader }
]
```

Performance Architecture

Frontend Performance

React Optimization
- Component memoization with `React.memo`
- State management with `useState` and `useReducer`

- Virtual scrolling for large student rosters
- Image optimization with Next.js Image component

Bundle Optimization
- Tree shaking for unused code elimination
- Dynamic imports for code splitting
- Static asset optimization and caching
- Service worker for offline capability

Backend Performance

Database Performance
- Connection pooling (max 20 connections)
- Query optimization with proper indexing
- Batch operations for bulk updates
- Read replicas for scaling read-heavy operations

Caching Strategy
- Redis for session data and active room state
- Browser caching for static assets
- API response caching for room metadata
- WebSocket connection state caching

Scalability Considerations

Horizontal Scaling
- Stateless API design for load balancing
- WebSocket connection sharing across instances
- Database sharding by room code prefix
- CDN for static asset delivery

Resource Management
- Memory-efficient data structures
- Garbage collection optimization
- Connection pooling and cleanup
- Background job processing for heavy operations

Monitoring and Observability

Metrics Collection

Application Metrics
- Active rooms and concurrent users
- Submission rates and approval times
- WebSocket connection stability
- API response times and error rates

Infrastructure Metrics
- Database query performance
- Memory and CPU utilization
- Network bandwidth usage
- Error rates and availability

Logging Strategy

Structured Logging

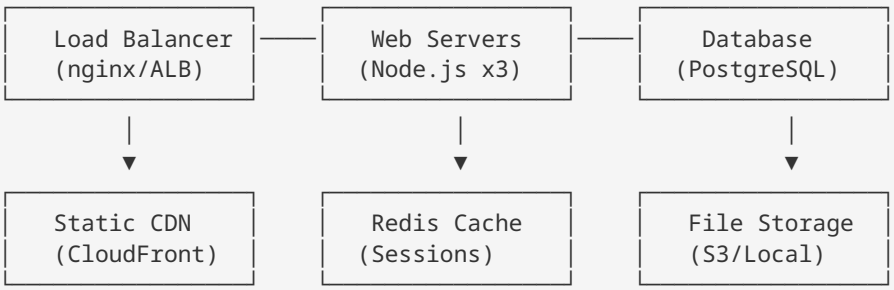
```
const logger = {
  info: (message: string, context: object) => {},
  warn: (message: string, context: object) => {},
  error: (message: string, error: Error, context: object) => {}
}

// Usage examples
logger.info('Room created', { roomCode, teacherId, studentCount })
logger.error('Submission failed', error, { roomCode, studentId })
```

Log Categories

- User actions: Room creation, submissions, approvals
- System events: Connections, disconnections, errors
- Performance: Query times, response latencies
- Security: Failed validations, rate limit hits

Deployment Architecture

Production Environment



Infrastructure Requirements

Minimum Production Setup

- 2 CPU cores, 4GB RAM per web server instance
- PostgreSQL with 2 CPU cores, 8GB RAM
- Redis with 1GB RAM for session storage
- Load balancer with SSL termination

Recommended Production Setup

- 3+ web server instances for high availability
- Database with read replicas and automated backups
- Redis cluster for session reliability
- Monitoring and alerting infrastructure

This architecture supports the PRD requirements while maintaining scalability, security, and performance for classroom environments with up to 50 concurrent rooms and 1,500+ simultaneous users.