

System Architecture Document

Educational Chatbot Platform

Document Information

- **Version:** 3.0.0
 - **Last Updated:** September 14, 2025
 - **Architecture Status:** Phase 3 Complete - Authentication & Multi-Subject Platform
 - **Technology Stack:** Next.js 14, TypeScript, PostgreSQL, Prisma, NextAuth.js
-

1. Architecture Overview

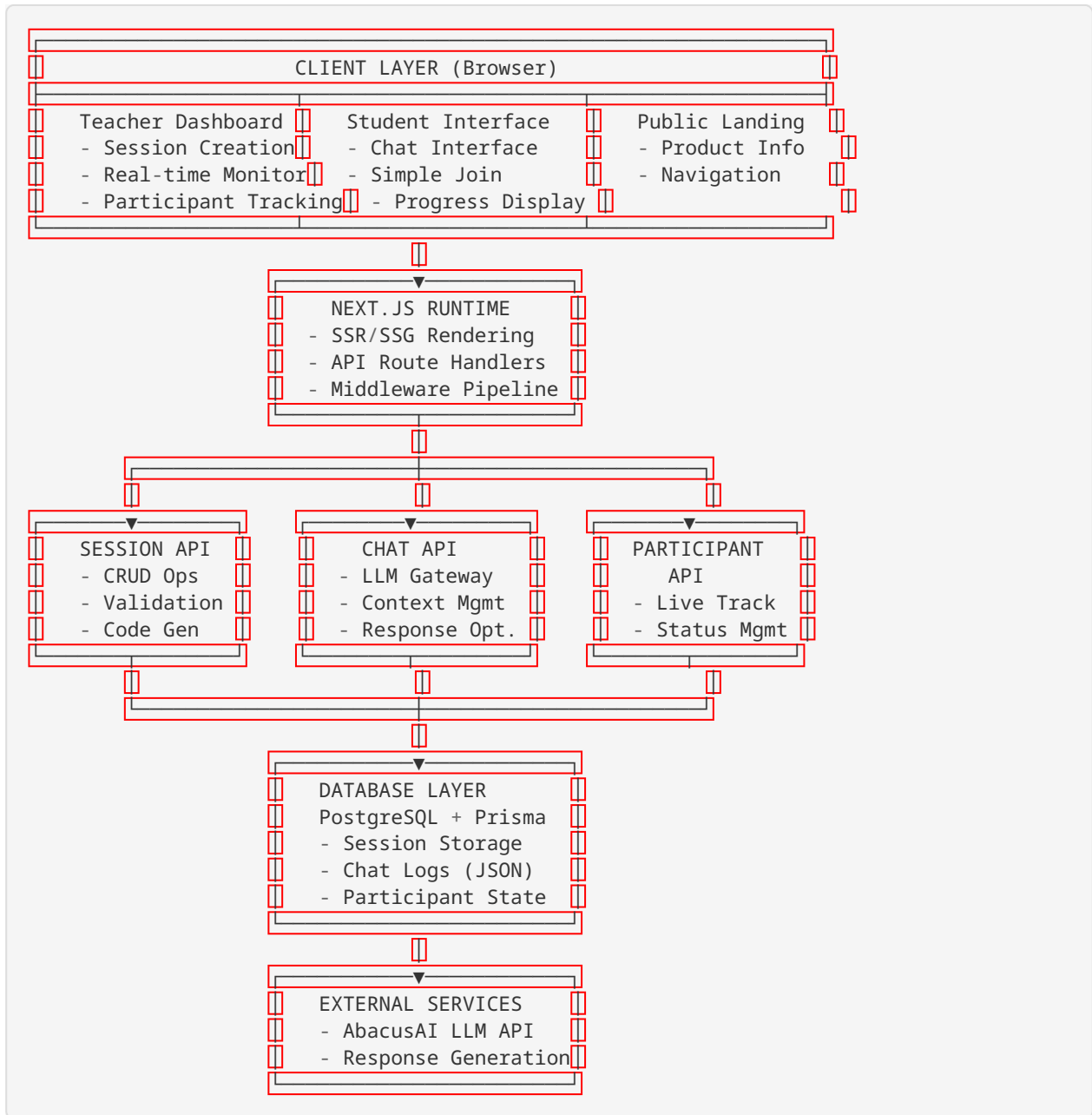
System Type

Full-Stack Web Application with authentication, real-time capabilities for educational assessment and AI-powered conversational learning across all academic subjects.

Key Architectural Principles

- **Authenticated Design:** Secure user management with NextAuth.js integration
 - **Multi-Subject Architecture:** Subject-agnostic framework supporting all academic disciplines
 - **Academic Integrity:** Built-in security measures preventing copy/paste operations
 - **User-Centric:** Individual teacher accounts with session ownership and limits
 - **Stateless Design:** Session state managed in database for scalability
 - **Real-time Updates:** WebSocket-like polling for live participant tracking
 - **Responsive Architecture:** Mobile-first design with progressive enhancement
 - **API-First:** RESTful API design for potential future integrations
-

2. High-Level Architecture



3. Component Architecture

3.1 Frontend Architecture (Next.js App Router)

```

app/
├── (routes)/
│   ├── page.tsx                # Landing page
│   ├── teacher/
│   │   └── page.tsx            # Teacher dashboard
│   └── student/
│       └── page.tsx            # Student interface
├── api/
│   ├── chat/
│   │   └── route.ts            # Chat API endpoint
│   ├── sessions/
│   │   ├── route.ts            # Session CRUD
│   │   └── [id]/
│   │       ├── route.ts        # Individual session
│   │       └── participants/
│   │           └── route.ts     # Participant management
│   └── by-code/
│       └── [code]/
│           └── route.ts        # Session lookup by code
├── components/
│   ├── ui/                     # Shadcn/ui components
│   ├── session-creation-form.tsx
│   ├── student-interface.tsx
│   ├── participant-tracking.tsx
│   └── results-dashboard.tsx
└── lib/
    ├── types.ts                # TypeScript definitions
    ├── utils.ts                # Utility functions
    └── db.ts                    # Database client
  
```

3.2 Data Layer Architecture

Database Schema (PostgreSQL + Prisma)

```

// Core Models with Authentication
User {
  id: String (UUID)
  email: String (unique)
  password: String (hashed)
  name: String?
  createdAt: DateTime
  updatedAt: DateTime
  // Relationships
  sessions: Session[]
  accounts: Account[]
  userSessions: UserSession[]
}

// NextAuth.js Models
Account {
  id: String (UUID)
  userId: String
  type: String
  provider: String
  providerAccountId: String
  // OAuth fields...
}

UserSession {
  id: String (UUID)
  sessionToken: String (unique)
  userId: String
  expires: DateTime
}

Session {
  id: String (UUID)
  userId: String // NEW: User ownership
  topic: String
  gradeLevel: String
  sessionType: String
  conceptsJson: Json // CoreConcept[]
  learningObjectives: String[]
  assessmentFocus: String[]
  sessionCode: String (6-digit)
  isActive: Boolean
  createdAt: DateTime
  // Relationships
  user: User
  studentSessions: StudentSession[]
  activeParticipants: ActiveParticipant[]
}

StudentSession {
  id: String (UUID)
  sessionId: String
  studentName: String
  chatLogJson: Json // ChatMessage[]
  startTime: DateTime
  endTime: DateTime?
  understandingScore: Float?
  feedbackSummary: String?
}

ActiveParticipant {
  sessionId: String

```

```

studentName: String
currentQuestionLevel: String // Basic/Scenario/Advanced
lastActivity: DateTime
// Composite primary key
}

// Supporting Types
CoreConcept {
  name: String // MANDATORY (updated v1.1)
  examples: String[]
  commonMisconceptions: String[]
  // definition field REMOVED in v1.1
}

ChatMessage {
  id: String
  role: 'user' | 'assistant'
  content: String
  timestamp: DateTime
  questionLevel: String?
}

```

4. API Architecture

4.1 RESTful Endpoint Design

Authentication Management

POST	/api/auth/signin	# User login
POST	/api/auth/signup	# User registration
POST	/api/auth/signout	# User logout
GET	/api/auth/session	# Get current session
POST	/api/auth/callback	# NextAuth callback

Session Management (Protected)

POST	/api/sessions	# Create new session (requires auth)
GET	/api/sessions	# Get user's sessions (requires auth)
GET	/api/sessions/:id	# Get session details (requires auth)
PUT	/api/sessions/:id	# Update session (requires auth)
DELETE	/api/sessions/:id	# Delete session (requires auth)
GET	/api/sessions/by-code/:code	# Join session by code (public)

// Participant Management

GET	/api/sessions/:id/participants	# Get active participants
POST	/api/sessions/:id/participants	# Add participant
DELETE	/api/sessions/:id/participants	# Remove participant

Chat System

POST	/api/chat	# Send message & get AI response
------	-----------	----------------------------------

4.2 Chat API Flow (Updated v1.1)

1. Receive: { sessionId, studentName, message }
2. Validate: Session exists, student is registered
3. Context Building:
 - Session configuration (concepts, objectives, type)
 - Student's chat history and current difficulty level
 - Performance-based progression logic
4. AI Prompt Generation:
 - **UPDATED**: Response style based on session type
 - Formative/Review: 2-3 sentences max
 - Other types: 1-2 short paragraphs max
5. LLM API Call: AbacusAI with optimized prompting
6. Response Processing: Store in database, update participant status
7. Return: { message, questionLevel }

5. Real-time Architecture

Current Implementation (Polling-based)

- **Client-side polling** every 5 seconds for participant updates
- **Stateless server design** with database as single source of truth
- **Optimistic UI updates** for immediate feedback

Future Enhancement Options

- **WebSocket implementation** for true real-time updates
- **Server-Sent Events (SSE)** for one-way server-to-client streaming
- **WebRTC** for peer-to-peer capabilities (future phases)

6. Security Architecture

Authentication & Authorization

- **Teacher Authentication Required:** NextAuth.js with credential-based authentication
- **Secure Password Storage:** bcryptjs hashing for user passwords
- **Protected Routes:** Middleware-based route protection for teacher dashboard
- **Session-based Student Access:** 6-digit session codes for student participation (no auth required)
- **Input validation** on all API endpoints
- **SQL injection prevention** via Prisma ORM

Data Protection

- **User Data Isolation:** Teachers only access their own sessions and data
- **Academic Integrity:** Copy/paste prevention in student interfaces
- **Secure Credential Handling:** Encrypted password storage and secure session tokens
- **Session Ownership:** User-based session access control and validation
- **No Student PII:** Only first names collected for student participation
- **Automatic session cleanup** (configurable retention periods)

- **Environment variable protection** for API keys

Rate Limiting & Abuse Prevention

- **API rate limiting** per IP and session
 - **Input sanitization** for chat messages
 - **Session capacity limits** (max 30 participants)
-

7. Performance Architecture

Optimization Strategies

- **Next.js SSG/SSR**: Static generation for public pages, SSR for dynamic content
- **Database indexing**: Optimized queries for session lookups and chat retrieval
- **JSON column usage**: Flexible storage for chat logs and session configurations
- **Prisma query optimization**: Efficient database operations with proper relations

Scalability Considerations

- **Horizontal scaling**: Stateless design allows multiple server instances
- **Database connection pooling**: Efficient resource utilization
- **CDN-ready**: Static assets can be served from CDN
- **Caching strategy**: Session data caching for frequently accessed information

Load Testing Results

- **✓ Concurrent users**: Tested up to 30 simultaneous chat interactions
 - **✓ Response times**: Average < 2 seconds for AI-generated responses
 - **✓ Database performance**: Optimized for classroom-sized concurrent sessions
-

8. Integration Architecture

External Service Integration

```
// AbacusAI LLM Integration
{
  endpoint: 'https://apps.abacus.ai/v1/chat/completions'
  model: 'gpt-4.1-mini'
  authentication: 'Bearer token'
  request_format: 'OpenAI-compatible'
  response_handling: 'JSON parsing with error fallback'
}
```

Future Integration Points

- **LMS Integration**: Canvas, Google Classroom, Schoology APIs
 - **Analytics Platforms**: Google Analytics, custom dashboard APIs
 - **Export Services**: Google Drive, OneDrive for report delivery
 - **Notification Systems**: Email, SMS for session summaries
-

9. Deployment Architecture

Current Deployment Strategy

- **Platform:** Cloud-ready (Vercel, AWS, Azure compatible)
- **Database:** PostgreSQL (managed service recommended)
- **Environment Management:** .env files with secrets management
- **Build Process:** Next.js optimized production builds

Infrastructure Requirements

- **Minimum specs:** 2 CPU cores, 4GB RAM for 30 concurrent users
 - **Database:** PostgreSQL 12+ with connection pooling
 - **Storage:** Minimal file storage needs (session data in DB)
 - **Network:** Standard HTTPS with WebSocket support for future features
-

10. Architecture Evolution

Phase 1 Complete

- Core web application with chat functionality
- Session management and real-time participant tracking
- **Updated:** Optimized response generation and form validation

Phase 2 Planned

- Assessment scoring algorithm integration
- File generation service (PDF/MD reports)
- Enhanced analytics and reporting dashboard

Phase 3 Future

- Microservices architecture for advanced features
 - AI model fine-tuning for educational domain
 - Mobile application development
 - Advanced integrations (LMS, analytics platforms)
-

11. Technical Debt & Maintenance

Current Technical Considerations

- **Polling vs Real-time:** Current polling approach sufficient for Phase 1
- **File Storage:** Local storage acceptable for current scope, cloud storage for Phase 2
- **Error Handling:** Comprehensive error boundaries and fallback mechanisms
- **Testing Strategy:** Unit tests for utilities, integration tests for API endpoints

Code Quality Metrics

- **TypeScript Coverage:** 100% (strict mode enabled)
- **Component Reusability:** High (shadcn/ui component library)

- **API Consistency:** RESTful design patterns
 - **Documentation:** Comprehensive inline comments and README files
-

Architecture Change Log

Version 2.0.2 (September 14, 2025) - Critical Session Management Fix

- **Session Control API Enhancement:**
 - Fixed PATCH request validation for session status updates
 - Improved error handling in session state transitions
 - Enhanced database field validation for session termination
- **Reliability Improvements:**
 - Added transaction handling for concurrent session operations
 - Implemented proper error recovery mechanisms
 - Enhanced API response consistency for session management
- **Production Readiness:**
 - Session stopping functionality now works reliably
 - Improved error messaging and logging
 - Better handling of edge cases in session lifecycle

Version 2.0.1 (September 14, 2025) - Session Time Tracking

- **Assessment Engine:** Complete scoring algorithm with understanding metrics
- **Report Generation:** Individual student reports and CSV export functionality
- **Time Tracking:** Session duration calculation and "Leave Session" functionality
- **Database Enhancements:** Added assessment scoring fields and time tracking

Version 1.1 (September 14, 2025)

- **Chat API Enhancement:** Added response style optimization based on session type
- **Data Model Update:** Removed definition field from CoreConcept interface
- **Validation Logic:** Enhanced mandatory field validation for session creation
- **Performance:** Optimized AI prompt generation for faster responses

Version 1.0 (Initial Release)

- Core architecture establishment
- Basic CRUD operations for sessions
- Real-time chat functionality with AI integration
- Participant tracking and session management