Technical Specification Document

Educational Chatbot Platform

Document Information

- **Version**: 1.1
- **Last Updated**: September 14, 2025
- **Implementation Status**: Phase 1 Complete
- **Development Team**: AI-Assisted Development

---

1. Technology Stack

Frontend

- **Framework**: Next.js 14.2.28 (App Router)
- **Language**: TypeScript 5.2.2
- **Styling**: Tailwind CSS 3.3.3 + shadcn/ui components
- **State Management**: React useState/useEffect + SWR for data fetching
- **UI Components**: Radix UI primitives with custom styling
- **Icons**: Lucide React 0.446.0

Backend

- **Runtime**: Node.js (Next.js API Routes)
- **Database**: PostgreSQL with Prisma ORM 6.7.0
- **Authentication**: Session-based (no user auth required)
- **External APIs**: AbacusAI LLM API (OpenAI-compatible)

Development Tools

- **Package Manager**: Yarn
- **Type Checking**: TypeScript strict mode
- **Code Quality**: ESLint 9.24.0 + Prettier
- **Build Tool**: Next.js built-in webpack 5.99.5

---

2. Database Schema Implementation

2.1 Prisma Schema Definition

```
// prisma/schema.prisma

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

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

model Session {
  id                   String               @id @default(cuid())
  topic                String
  gradeLevel           String
  sessionType          String
  conceptsJson         Json                 // CoreConcept[]
  learningObjectives   String[]
  assessmentFocus      String[]
  difficultyProgression String?
  additionalContext    String?
  sessionCode          String               @unique
  isActive             Boolean              @default(true)
  createdAt            DateTime             @default(now())
  startTime            DateTime?
  endTime              DateTime?

  // Relations
  studentSessions      StudentSession[]
  activeParticipants   ActiveParticipant[]

  @@map("sessions")
}

model StudentSession {
  id                   String    @id @default(cuid())
  sessionId            String
  studentName          String
  chatLogJson          Json      @default("[]") // ChatMessage[]
  startTime            DateTime  @default(now())
  endTime              DateTime?
  understandingScore   Float?
  feedbackSummary      String?

  // Relations
  session              Session   @relation(fields: [sessionId], references: [id], onDe
lete: Cascade)

  @@unique([sessionId, studentName])
  @@map("student_sessions")
}

model ActiveParticipant {
  id                   String   @id @default(cuid())
  sessionId            String
  studentName          String
  currentQuestionLevel String   @default("Basic") // Basic|Scenario|Advanced
  lastActivity         DateTime @default(now())

  // Relations
  session              Session  @relation(fields: [sessionId], references: [id], onDel
ete: Cascade)
```

```
  @@unique([sessionId, studentName], name: "sessionId_studentName")
  @@map("active_participants")
}
```

2.2 Data Models (TypeScript)

```typescript
// lib/types.ts

export interface SessionFormData {
  topic: string;
  gradeLevel: string;
  sessionType: string;
  concepts: CoreConcept[];
  learningObjectives: string[];
  assessmentFocus: string[];
  difficultyProgression: string;
  additionalContext?: string;
}

// UPDATED v1.1: Removed definition field
export interface CoreConcept {
  name: string;                    // MANDATORY
  examples: string[];
  commonMisconceptions: string[];
}

export interface ChatMessage {
  id: string;
  role: 'user' | 'assistant';
  content: string;
  timestamp: Date;
  questionLevel?: string;
}

// Constants
export const GRADE_LEVELS = [
  '9th Grade', '10th Grade', '11th Grade', '12th Grade'
] as const;

export const SESSION_TYPES = [
  'Pre-Assessment', 'Formative Check', 'Review Session',
  'Unit Assessment', 'Final Review'
] as const;

export const ASSESSMENT_FOCUS_AREAS = [
  'Vocabulary Understanding', 'Concept Application', 'Critical Thinking',
  'Problem Solving', 'Case Study Analysis', 'Real-world Connections'
] as const;
```

3. API Implementation Details

3.1 Chat API Endpoint (`/api/chat/route.ts`)

```
export async function POST(request: NextRequest) {
  try {
    const { sessionId, studentName, message } = await request.json();

    // Session & Student Validation
    const session = await prisma.session.findUnique({
      where: { id: sessionId }
    });

    if (!session) {
      return new Response('Session not found', { status: 404 });
    }

    // Progressive Difficulty Logic
    const participant = await prisma.activeParticipant.findFirst({
      where: { sessionId, studentName }
    });

    const currentLevel = participant?.currentQuestionLevel || 'Basic';
    const conversationLength = /* chat history length calculation */;

    let nextLevel = currentLevel;
    if (currentLevel === 'Basic' && conversationLength >= 2) {
      nextLevel = 'Scenario';
    } else if (currentLevel === 'Scenario' && conversationLength >= 4) {
      nextLevel = 'Advanced';
    }

    // UPDATED v1.1: Response Style Optimization
    let responseStyle = '';
    if (session.sessionType === 'Formative Assessment' ||
        session.sessionType === 'Review Session') {
      responseStyle = `
RESPONSE STYLE: Keep responses BRIEF and DIRECT for ${session.sessionType}:
- Maximum 2-3 sentences
- No lengthy explanations unless student asks for clarification
- Focus on quick check-ins and targeted questions
- Get straight to the point`;
    } else {
      responseStyle = `
RESPONSE STYLE: Keep responses concise but informative:
- 1-2 short paragraphs maximum
- Clear and focused explanations`;
    }

    // LLM API Integration
    const response = await fetch('https://apps.abacus.ai/v1/chat/completions', {
      method: 'POST',
      headers: {
        'Content-Type': 'application/json',
        'Authorization': `Bearer ${process.env.ABACUSAI_API_KEY}`
      },
      body: JSON.stringify({
        model: 'gpt-4.1-mini',
        messages: [
          { role: 'system', content: systemMessage },
          { role: 'user', content: message }
        ],
        stream: false,
        max_tokens: 1000,
        temperature: 0.7
      })
```

```
    });

    // Response Processing & Database Storage
    const data = await response.json();
    const assistantMessage = data.choices?.[0]?.message?.content ||
                             'I apologize, but I encountered an issue...';

    // Update chat log and participant status
    // ... database update logic

    return new Response(JSON.stringify({
      message: assistantMessage,
      questionLevel: nextLevel
    }), {
      headers: { 'Content-Type': 'application/json' }
    });

  } catch (error) {
    console.error('Error in chat API:', error);
    return new Response('Internal server error', { status: 500 });
  }
}
```

3.2 Session Management API (`/api/sessions/route.ts`)

```
// Create Session
export async function POST(request: NextRequest) {
  try {
    const formData = await request.json();

    // UPDATED v1.1: Enhanced validation
    if (!formData.topic?.trim() || !formData.gradeLevel || !formData.sessionType) {
      return new Response('Missing required fields', { status: 400 });
    }

    if (formData.concepts.some((c: any) => !c.name?.trim())) {
      return new Response('All concepts must have names', { status: 400 });
    }

    if (formData.learningObjectives.some((obj: string) => !obj.trim())) {
      return new Response('All learning objectives must be filled', { status: 400 });
    }

    // Generate unique session code
    const sessionCode = Math.random().toString(36).substring(2, 8).toUpperCase();

    const session = await prisma.session.create({
      data: {
        topic: formData.topic,
        gradeLevel: formData.gradeLevel,
        sessionType: formData.sessionType,
        conceptsJson: formData.concepts,
        learningObjectives: formData.learningObjectives,
        assessmentFocus: formData.assessmentFocus,
        difficultyProgression: formData.difficultyProgression,
        additionalContext: formData.additionalContext,
        sessionCode,
        startTime: new Date()
      }
    });

    return NextResponse.json(session);
  } catch (error) {
    console.error('Session creation error:', error);
    return new Response('Session creation failed', { status: 500 });
  }
}

// Get Sessions with Statistics
export async function GET() {
  try {
    const sessions = await prisma.session.findMany({
      include: {
        _count: {
          select: {
            studentSessions: true,
            activeParticipants: true
          }
        }
      },
      orderBy: { createdAt: 'desc' }
    });

    const sessionsWithStats = sessions.map(session => ({
      ...session,
      participantCount: session._count.activeParticipants,
      totalStudents: session._count.studentSessions
```

```
    }));

    return NextResponse.json(sessionsWithStats);
  } catch (error) {
    return new Response('Failed to fetch sessions', { status: 500 });
  }
}
```

---

4. Frontend Component Architecture

4.1 Session Creation Form (Updated v1.1)

```
// components/session-creation-form.tsx

export function SessionCreationForm({ onSessionCreated }: SessionCreationFormProps) {
  const [formData, setFormData] = useState<SessionFormData>({
    topic: '',
    gradeLevel: '',
    sessionType: '',
    concepts: [{ name: '', examples: [''], commonMisconceptions: [''] }], // No defini-
tion field
    learningObjectives: [''],
    assessmentFocus: [],
    difficultyProgression: '',
    additionalContext: ''
  });

  // UPDATED v1.1: Enhanced validation
  const handleSubmit = async (e: React.FormEvent) => {
    e.preventDefault();

    // Mandatory field validation
    if (!formData.topic.trim() || !formData.gradeLevel || !formData.sessionType) {
      toast({
        title: "Missing Required Fields",
        description: "Please fill in topic, grade level, and session type.",
        variant: "destructive"
      });
      return;
    }

    if (formData.concepts.some(c => !c.name.trim())) {
      toast({
        title: "Missing Concept Names",
        description: "Please ensure all concepts have names.",
        variant: "destructive"
      });
      return;
    }

    if (formData.learningObjectives.some(obj => !obj.trim())) {
      toast({
        title: "Missing Learning Objectives",
        description: "Please ensure all learning objectives are filled in.",
        variant: "destructive"
      });
      return;
    }

    // ... form submission logic
  };

  return (
    <form onSubmit={handleSubmit} className="space-y-8">
      {/* Core Concepts Section - UPDATED */}
      <Card>
        <CardHeader>
          <CardTitle>Core Concepts (2-4) *</CardTitle>
          <CardDescription>
            Define the key concepts students should understand
          </CardDescription>
        </CardHeader>
        <CardContent>
          {formData.concepts.map((concept, conceptIndex) => (
```

```
            <div key={conceptIndex} className="border rounded-lg p-4 space-y-4">
              {/* REMOVED: Definition field */}
              <div>
                <Label>Concept Name *</Label>
                <Input
                  value={concept.name}
                  onChange={(e) => updateConcept(conceptIndex, 'name', e.target.value)}
                  placeholder="e.g., Market Segmentation"
                  required
                />
              </div>

              {/* Examples and Misconceptions sections remain */}
            </div>
          ))}
        </CardContent>
      </Card>

      {/* Learning Objectives - UPDATED */}
      <Card>
        <CardHeader>
          <CardTitle>Learning Objectives (2-3) *</CardTitle>
        </CardHeader>
        <CardContent>
          {formData.learningObjectives.map((objective, index) => (
            <Input
              key={index}
              value={objective}
              onChange={(e) => updateObjective(index, e.target.value)}
              placeholder={`Learning objective ${index + 1}... *`}
              required
            />
          ))}
        </CardContent>
      </Card>
    </form>
  );
}
```

4.2 Student Chat Interface

```tsx
// components/student-interface.tsx

export function StudentInterface() {
  const [messages, setMessages] = useState<ChatMessage[]>([]);
  const [currentMessage, setCurrentMessage] = useState('');
  const [isLoading, setIsLoading] = useState(false);
  const [questionLevel, setQuestionLevel] = useState('Basic');

  const sendMessage = async () => {
    if (!currentMessage.trim() || isLoading) return;

    setIsLoading(true);
    const userMessage: ChatMessage = {
      id: Date.now().toString(),
      role: 'user',
      content: currentMessage,
      timestamp: new Date(),
      questionLevel
    };

    setMessages(prev => [...prev, userMessage]);
    setCurrentMessage('');

    try {
      const response = await fetch('/api/chat', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({
          sessionId: session?.id,
          studentName: studentName,
          message: currentMessage
        })
      });

      if (!response.ok) throw new Error('Failed to send message');

      const data = await response.json();

      const assistantMessage: ChatMessage = {
        id: (Date.now() + 1).toString(),
        role: 'assistant',
        content: data.message,
        timestamp: new Date(),
        questionLevel: data.questionLevel
      };

      setMessages(prev => [...prev, assistantMessage]);
      setQuestionLevel(data.questionLevel);

    } catch (error) {
      console.error('Error sending message:', error);
      // Error handling with fallback message
    } finally {
      setIsLoading(false);
    }
  };

  return (
    <div className="flex flex-col h-full">
      {/* Chat Messages */}
      <ScrollArea className="flex-1 p-4">
        {messages.map((message) => (
```

```
            <div key={message.id} className={`mb-4 ${
              message.role === 'user' ? 'text-right' : 'text-left'
            }`}>
              <div className={`inline-block max-w-[80%] p-3 rounded-lg ${
                message.role === 'user'
                  ? 'bg-blue-600 text-white'
                  : 'bg-gray-200 text-gray-900'
              }`}>
                {message.content}
              </div>
            </div>
          ))}
          {isLoading && <div className="text-center">AI is thinking...</div>}
        </ScrollArea>

        {/* Message Input */}
        <div className="border-t p-4">
          <div className="flex space-x-2">
            <Input
              value={currentMessage}
              onChange={(e) => setCurrentMessage(e.target.value)}
              onKeyPress={(e) => e.key === 'Enter' && sendMessage()}
              placeholder="Type your response..."
              disabled={isLoading}
            />
            <Button onClick={sendMessage} disabled={isLoading}>
              Send
            </Button>
          </div>
        </div>
      </div>
    );
}
```

---

5. Environment Configuration

5.1 Environment Variables

```
# .env.local
DATABASE_URL="postgresql://username:password@localhost:5432/educational_chatbot"
ABACUSAI_API_KEY="your_api_key_here"
NEXTAUTH_URL="http://localhost:3000"
NEXTAUTH_SECRET="your_secret_key_here"
```

5.2 Package Configuration

```
// package.json (key dependencies)
{
  "name": "educational-chatbot",
  "version": "1.1.0",
  "scripts": {
    "dev": "next dev",
    "build": "next build",
    "start": "next start",
    "db:push": "prisma db push",
    "db:studio": "prisma studio",
    "db:generate": "prisma generate"
  },
  "dependencies": {
    "next": "14.2.28",
    "react": "18.2.0",
    "@prisma/client": "6.7.0",
    "@radix-ui/react-*": "latest",
    "tailwindcss": "3.3.3",
    "typescript": "5.2.2",
    "zod": "3.23.8"
  },
  "devDependencies": {
    "prisma": "6.7.0",
    "@types/node": "20.6.2",
    "@types/react": "18.2.22",
    "eslint": "9.24.0"
  }
}
```

---

6. Performance Specifications

6.1 Response Time Requirements

- **Chat API**: < 2 seconds (including LLM processing)
- **Session Creation**: < 500ms
- **Participant Updates**: < 1 second
- **Database Queries**: < 100ms (optimized with indexes)

6.2 Concurrent User Support

- **Target**: 20-30 students per session
- **Maximum**: 50 concurrent sessions system-wide
- **Database Connections**: Pool size of 20 connections
- **Memory Usage**: < 512MB per session (estimated)

6.3 Database Optimization

```
-- Index optimizations
CREATE INDEX idx_sessions_code ON sessions(session_code);
CREATE INDEX idx_sessions_active ON sessions(is_active);
CREATE INDEX idx_student_sessions_composite ON student_sessions(session_id, stu-
dent_name);
CREATE INDEX idx_active_participants_session ON active_participants(session_id);
CREATE INDEX idx_active_participants_activity ON active_participants(last_activity);
```

---

7. Error Handling & Logging

7.1 API Error Responses

```
// Standardized error response format
interface APIError {
  error: string;
  message: string;
  statusCode: number;
  timestamp: string;
}

// Error handling middleware
export function handleAPIError(error: unknown, request: NextRequest) {
  console.error('API Error:', error);

  if (error instanceof ValidationError) {
    return new Response(JSON.stringify({
      error: 'Validation Failed',
      message: error.message,
      statusCode: 400,
      timestamp: new Date().toISOString()
    }), { status: 400 });
  }

  return new Response(JSON.stringify({
    error: 'Internal Server Error',
    message: 'An unexpected error occurred',
    statusCode: 500,
    timestamp: new Date().toISOString()
  }), { status: 500 });
}
```

7.2 Client-side Error Boundaries

```
// components/error-boundary.tsx
export class ErrorBoundary extends React.Component {
  constructor(props: any) {
    super(props);
    this.state = { hasError: false };
  }

  static getDerivedStateFromError(error: Error) {
    return { hasError: true };
  }

  componentDidCatch(error: Error, errorInfo: React.ErrorInfo) {
    console.error('React Error Boundary:', error, errorInfo);
  }

  render() {
    if (this.state.hasError) {
      return (
        <div className="text-center p-8">
          <h2 className="text-xl font-bold text-red-600 mb-4">
            Something went wrong
          </h2>
          <Button onClick={() => this.setState({ hasError: false })}>
            Try Again
          </Button>
        </div>
      );
    }

    return this.props.children;
  }
}
```

8. Testing Strategy

8.1 Unit Testing (Jest + React Testing Library)

```
// __tests__/api/chat.test.ts
import { POST } from '@/app/api/chat/route';
import { createMocks } from 'node-mocks-http';

describe('/api/chat', () => {
  it('should return AI response for valid input', async () => {
    const { req } = createMocks({
      method: 'POST',
      body: {
        sessionId: 'test-session',
        studentName: 'John',
        message: 'What is marketing?'
      }
    });

    const response = await POST(req as any);
    const data = await response.json();

    expect(response.status).toBe(200);
    expect(data).toHaveProperty('message');
    expect(data).toHaveProperty('questionLevel');
  });
});
```

8.2 Integration Testing

```
// __tests__/session-flow.test.ts
describe('Session Flow Integration', () => {
  it('should create session, join as student, and send message', async () => {
    // 1. Create session via API
    const sessionResponse = await fetch('/api/sessions', {
      method: 'POST',
      body: JSON.stringify(mockSessionData)
    });
    const session = await sessionResponse.json();

    // 2. Join session as student
    const joinResponse = await fetch(`/api/sessions/by-code/${session.sessionCode}`);
    expect(joinResponse.status).toBe(200);

    // 3. Send chat message
    const chatResponse = await fetch('/api/chat', {
      method: 'POST',
      body: JSON.stringify({
        sessionId: session.id,
        studentName: 'TestStudent',
        message: 'Hello'
      })
    });
    expect(chatResponse.status).toBe(200);
  });
});
```

9. Deployment Specifications

9.1 Build Process

```
# Production build commands
yarn install --frozen-lockfile
npx prisma generate
yarn build
yarn start
```

9.2 Environment Requirements

- **Node.js**: 18.0+
- **PostgreSQL**: 12.0+
- **Memory**: 2GB minimum, 4GB recommended
- **CPU**: 2 cores minimum for 30 concurrent users
- **Storage**: 10GB (for logs and session data)

9.3 Health Check Endpoint

```
// app/api/health/route.ts
export async function GET() {
  try {
    // Check database connection
    await prisma.$queryRaw`SELECT 1`;

    // Check external API connectivity
    const testResponse = await fetch(process.env.ABACUSAI_API_ENDPOINT!, {
      headers: { 'Authorization': `Bearer ${process.env.ABACUSAI_API_KEY}` }
    });

    return NextResponse.json({
      status: 'healthy',
      timestamp: new Date().toISOString(),
      database: 'connected',
      externalAPI: testResponse.ok ? 'connected' : 'degraded'
    });
  } catch (error) {
    return NextResponse.json({
      status: 'unhealthy',
      error: error instanceof Error ? error.message : 'Unknown error'
    }, { status: 500 });
  }
}
```

---

10. Technical Debt & Future Improvements

10.1 Current Limitations

- **Real-time Updates**: Using polling instead of WebSockets
- **File Storage**: No cloud storage integration yet
- **Caching**: No Redis implementation for session caching
- **Monitoring**: Basic error logging, no advanced observability

10.2 Phase 2 Technical Requirements

- **Assessment Engine**: Scoring algorithm implementation
- **File Generation**: PDF/Markdown report creation
- **Cloud Storage**: AWS S3 or similar for file storage
- **Advanced Analytics**: Chart.js integration for data visualization

10.3 Security Enhancements

- **Rate Limiting**: Implement per-IP and per-session limits
- **Input Sanitization**: Enhanced validation for chat messages
- **Session Encryption**: Encrypt sensitive session data at rest
- **CORS Configuration**: Proper origin restrictions for production

---

Technical Change Log

Version 1.1 (September 14, 2025)

Backend Changes:
- ✅ Updated Chat API to optimize response style based on session type
- ✅ Enhanced LLM prompt generation with concise response instructions
- ✅ Improved session creation validation for mandatory fields

Frontend Changes:
- ✅ Removed definition field from CoreConcept interface and UI
- ✅ Added mandatory field validation with user-friendly error messages
- ✅ Updated form labels to indicate required fields with asterisks

Database Changes:
- ✅ Updated CoreConcept type definition (removed definition field)
- ✅ Enhanced validation constraints for concept names and learning objectives

Performance Improvements:
- ✅ Optimized AI response generation for faster processing
- ✅ Reduced response verbosity for improved user experience

Version 1.0 (Initial Implementation)

- ✅ Core application architecture established
- ✅ Full-stack implementation with Next.js and PostgreSQL
- ✅ AI chat integration with AbacusAI
- ✅ Real-time participant tracking
- ✅ Responsive UI with shadcn/ui components