TECH STACK DOCUMENT

Bloom Academia - 30-Day MVP

Complete Technology Stack & Architecture

INTRODUCTION

This document outlines every technology, library, service, and tool we're using to build the Bloom Academia MVP in 30 days. Each choice has been made to maximize development speed while maintaining production quality for the hackathon demo and investor presentations.

Key Principles:

- Serverless architecture for zero DevOps overhead
- Modern, well-documented technologies with strong community support
- Gemini 3 as the core (hackathon requirement)
- Fast iteration and deployment
- Production-ready from day one

1. COMPLETE STACK OVERVIEW

Layer	Technology	Purpose
FRONTEND		
Framework	Next.js 15	React framework with SSR/SSG
UI Library	React 18	Core UI library
Language	TypeScript	Type safety, better DX
Styling	Tailwind CSS	Utility-first CSS framework
Components	shadcn/ui	Pre-built accessible components
Animation	Framer Motion	Smooth animations & transitions
Canvas	Fabric.js / Konva	Whiteboard rendering & SVG
State Management	Zustand	Lightweight global state
BACKEND		
API Layer	Next.js API Routes	Serverless functions
Database	Supabase (PostgreSQL)	Managed database + auth
ORM/Client	Supabase JS Client	Database interactions
Real-time	WebSocket	Bidirectional voice streaming
AI / ML		
AI Model	Gemini 3 Flash	Core AI teaching engine
Speech-To-Text	Soniox	Real-time audio transcription via WebSocket
Text-To-Speech	Google cloud TTS	High-quality voice synthesis (Neural2 voices, streaming audio generation)
Memory System	Custom 3 layer architecture	Personalized learning context (User Profile + Session Memory + Long-term Analysis)
SVG generation	Gemini 3 Flash	On-the-fly visual aids generated during teaching responses

INFRASTRUCTURE		
Hosting	Vercel	Deploy + CDN + serverless
Version Control	Git + GitHub	Code repository
Domain/DNS	Vercel Domains	Custom domain setup
DEV TOOLS		
Code Editor	VS Code / Cursor	IDE with AI assistance
Package Manager	pnpm / npm	Dependency management
Linting	ESLint + Prettier	Code quality & formatting

2. FRONTEND TECHNOLOGIES

2.1 Next.js 15 (Core Framework)

Why Next.js:

- React framework with built-in backend capabilities (API routes)
- Server-side rendering (SSR) for fast initial loads
- Static site generation (SSG) for landing page performance
- Built-in routing (no react-router needed)
- Optimized production builds out of the box
- Perfect Vercel integration (same company)

What we use it for:

- Landing page (SSG for speed)
- Application shell and routing
- API routes for backend logic
- Image optimization
- Font optimization

Installation:

```
npx create-next-app@latest ai-school --typescript --tailwind --app
```

2.2 React 18

Why React:

- Component-based architecture (perfect for reusable UI)
- Huge ecosystem and community
- Hooks for state management
- Concurrent features for smooth UX

Key React features we use:

- useState, useEffect for component state
- useContext for global state (with Zustand)
- useRef for canvas and audio element references
- Custom hooks for reusable logic

2.3 TypeScript

Why TypeScript:

- Catch errors before runtime
- Better autocomplete in VS Code/Cursor
- Self-documenting code (types as documentation)
- Easier refactoring
- Industry standard for production apps

Configuration:
tsconfig.json comes pre-configured with Next.js

2.4 Tailwind CSS

Why Tailwind:

- Utility-first CSS = fast styling
- No CSS file management
- Built-in design system (colors, spacing, etc.)
- Responsive design made easy
- Tree-shaking = minimal production CSS

Installation:
Comes with Next.js setup, configured automatically

Custom config (tailwind.config.js):
```
theme: {
  extend: {
    colors: {
      primary: '#4A90E2',
      secondary: '#7ED321',
      accent: '#F5A623'
    }
  }
}
```

2.5 shadcn/ui

Why shadcn/ui:

- Pre-built accessible components
- Built on Radix UI (accessibility primitives)
- Styled with Tailwind
- Copy-paste components (not a package dependency)
- Customizable source code

Components we'll use:

- Button
- Dialog/Modal
- Card
- Progress
- Dropdown Menu

Installation:
```
npx shadcn-ui@latest init
npx shadcn-ui@latest add button card progress
```

2.6 Framer Motion

Why Framer Motion:

- Declarative animations for React
- Smooth, performant animations
- Easy gesture support
- Great documentation

What we use it for:

- Page transitions
- Whiteboard content animations (fade, slide)
- Voice indicator pulsing
- Celebration effects (confetti trigger)

Installation:

```
npm install framer-motion
```

Example usage:

```
import { motion } from 'framer-motion'
<motion.div initial={{opacity: 0}} animate={{opacity: 1}} />
```

2.7 Canvas Libraries (Fabric.js OR Konva)

Purpose: Interactive whiteboard rendering

Option 1: Fabric.js

- Powerful canvas library
- Great for SVG manipulation
- Good documentation
- Installation: npm install fabric

Option 2: Konva (with react-konva)

- React-first approach
- Declarative API
- Easier React integration
- Installation: npm install konva react-konva

Recommendation: Start with react-konva
Better React integration, easier to work with. Can switch to Fabric.js if needed.

What we'll render:

- SVG diagrams (pizzas, shapes, number lines)
- Mathematical equations
- Text with formatting
- Animated drawings

2.8 State Management (Zustand)

Why Zustand:

- Lightweight (< 1KB)
- Simpler than Redux
- TypeScript-friendly
- No boilerplate
- Perfect for small-to-medium apps

Installation:

```
npm install zustand
```

What we store:

- User profile (name, age, grade)
- Current lesson state
- Voice connection status
- Loading states
- Error messages

Example store:

```
import create from 'zustand'

const useStore = create((set) => ({
  userName: '',
  setUserName: (name) => set({ userName: name }),
  lessonProgress: 0,
  updateProgress: (progress) => set({ lessonProgress: progress })
}))
```

3. BACKEND TECHNOLOGIES

3.1 Next.js API Routes (Serverless Backend)

What are API Routes:

- Server-side code that runs on Vercel's edge network
- Located in `/app/api/` directory (Next.js 15 App Router)
- Each file = an API endpoint
- Automatically deployed with your frontend

Why this approach:

- No separate backend server to manage
- Auto-scaling (serverless)
- Same codebase as frontend
- TypeScript support
- Easy authentication
- Built-in Request/Response Web APIs

API Routes we'll create:

1. `/api/teach` (Main Teaching Endpoint)

- POST endpoint for complete teaching interaction
- Receives transcribed text from frontend
- Builds AI context from 3-layer memory system
- Calls Gemini 3 Flash for teaching response
- Generates audio via Google Cloud TTS
- Returns: text response, SVG code, audio base64
- Saves interaction to session memory

2. `/api/stt/temp-key` (Soniox Authentication)

- GET endpoint to generate temporary Soniox API key
- Keeps permanent API key secret from frontend
- Returns short-lived key for client-side WebSocket connection
- Expires in 60 seconds (renewable)

3. `/api/tts/synthesize` (Text-to-Speech)

- POST endpoint for audio generation
- Receives text, returns audio buffer
- Uses Google Cloud TTS Neural2 voices
- Handles streaming synthesis

4. `/api/memory/profile` (User Profile Management)

- GET: Fetch user learning profile from Supabase
- POST: Update user profile with discovered insights
- Returns: learning style, strengths, struggles, preferences

5. `/api/memory/context` (Context Builder)

- POST endpoint to build AI context
- Combines: user profile + session history + lesson data
- Returns: formatted system prompt for Gemini

6. `/api/memory/analyze` (Session Analysis)

- POST endpoint called after session ends
- Uses Gemini to analyze learning patterns
- Updates user profile with new insights

7. `/api/progress/save` (Progress Tracking)

- POST endpoint to save lesson progress
- Writes to Supabase: mastery level, time spent, completed status

8. `/api/progress/load` (Progress Loading)

- GET endpoint to load user progress
- Reads from Supabase, returns progress data

9. `/api/lessons` (Lesson Management)

- GET: List all available lessons
- Returns: lesson metadata, difficulty, subject

10. `/api/lessons/[id]` (Lesson Details)

- GET: Fetch specific lesson content
- Returns: full lesson structure and learning objectives

11. `/api/sessions/start` (Session Management)

- POST: Create new learning session in database
- Returns: session ID for tracking interactions

12. `/api/sessions/end` (Session Completion)

- POST: Mark session as ended

- Triggers learning analysis
- Updates user profile

3.2 Supabase (Database + BaaS)

What is Supabase:

- Open-source Firebase alternative
- PostgreSQL database (cloud-hosted)
- Built-in authentication
- Real-time subscriptions
- Auto-generated REST API
- Row-level security

Why Supabase:

- No database management (fully managed)
- Free tier is generous (500MB database, 2GB bandwidth)
- Easy to use JavaScript client
- Real-time features (for future use)
- Auth ready when we need it (post-MVP)

Setup:

1. Sign up at supabase.com
2. Create new project
3. Get API keys (anon key + service role key)
4. Install client: npm install @supabase/supabase-js

Database tables we'll create:

users:
- id, name, age, grade_level, learning_style, created_at

lessons:
- id, title, subject, description, grade_level

progress:
- id, user_id, lesson_id, mastery_level, time_spent, completed

sessions:
- id, user_id, lesson_id, started_at, ended_at, interaction_count

Client initialization:

```
// lib/supabase.ts
import { createClient } from '@supabase/supabase-js'

export const supabase = createClient(
  process.env.NEXT_PUBLIC_SUPABASE_URL!,
  process.env.NEXT_PUBLIC_SUPABASE_ANON_KEY!
)
```

3.3 WebSocket (Real-time Communication)

Purpose:

- Bidirectional audio streaming with Gemini Live API
- Low-latency voice communication

Implementation:

- Native WebSocket API (browser built-in)
- Next.js API route acts as WebSocket proxy
- Connects frontend → Next.js backend → Gemini API

Why proxy through backend:

- Hide Gemini API key from frontend
- Add rate limiting if needed
- Log conversations for debugging
- Handle reconnection logic

4. AI / ML TECHNOLOGIES

4.1 Gemini 3 Flash

Model Details:

- **Model name:** `gemini-3-flash-preview`
- **Released:** January 2025
- **Context window:** 1 million tokens
- **Input modalities:** text, images, video, audio, PDFs
- **Output modalities:** text only (no native audio output)
- **Advanced reasoning:** Thinking levels (minimal, low, medium, high)
- **Elo Score:** 1501 (state-of-the-art reasoning)

Why Gemini 3 Flash:

- Hackathon requirement (Gemini 3 hackathon)
- Superior reasoning capability for complex teaching scenarios
- Best-in-class multimodal understanding
- 1M token context window (maintains full conversation history)
- Can generate SVG code inline for visual aids
- Excellent at explaining complex topics simply
- Cost-effective at Flash pricing tier

API Access:

- Sign up at ai.google.dev
- Get API key from Google AI Studio
- Apply for hackathon credits (free tier available for Flash)
- Install SDK: `npm install @google/genai`

Pricing:

- **Input:** $0.50 per 1M tokens
- **Output:** $3.00 per 1M tokens
- **Free tier available** for `gemini-3-flash-preview`
- Estimated cost per 30-min lesson: ~$0.05-0.10

4.2 Soniox (Speech-to-Text)

What is Soniox:

- Real-time speech recognition via WebSocket
- Ultra-low latency transcription (~100-300ms)
- 60+ languages supported
- Speaker diarization and endpoint detection

- Browser-native integration (no server proxying needed)

Key Features:

- **Endpoint detection:** Knows when user stops speaking
- **Interim results:** Real-time word-by-word transcription
- **High accuracy:** Trained on diverse accents and backgrounds
- **Language identification:** Auto-detect spoken language
- **Context customization:** Improve accuracy with domain-specific terms

API Access:

- Sign up at soniox.com
- Get API key
- Generate temporary keys for client-side usage (60-second expiry)
- Install SDK: `npm install @soniox/speech-to-text-web`

Pricing:

- Real-time API: Contact for pricing
- Typical cost: ~$0.005-0.01 per minute
- Estimated cost per 30-min lesson: ~$0.15-0.30

4.3 Web Audio API (Browser Audio)

What is Web Audio API:

- Browser built-in API for audio processing
- No installation needed
- Captures microphone input
- Plays audio output

What we use it for:

- Request microphone permission
- Capture student's voice
- Convert to audio chunks for streaming
- Play AI's voice responses
- Show audio visualization (voice indicator)

Key APIs:

- navigator.mediaDevices.getUserMedia() - get mic access
- AudioContext - audio processing
- MediaRecorder - record audio chunks
- Audio element - play responses

Browser support:

- Chrome/Edge: Full support
- Safari: Full support (requires user gesture)
- Firefox: Full support

5. INFRASTRUCTURE & DEPLOYMENT

5.1 Vercel (Hosting & Deployment)

What is Vercel:

- Serverless hosting platform
- Made by the creators of Next.js
- Global CDN for fast content delivery
- Automatic deployments from GitHub
- Built-in preview deployments

Why Vercel:

- Perfect Next.js integration
- Zero-config deployments
- Automatic HTTPS
- Edge functions (serverless API routes)
- Free tier is generous (100GB bandwidth)
- Custom domains included

Deployment workflow:

5. Push code to GitHub main branch
6. Vercel automatically detects changes
7. Builds and deploys in ~2 minutes
8. Live at your-app.vercel.app
9. Every PR gets a preview URL

Setup:

10. Sign up at vercel.com
11. Connect GitHub repository
12. Import Next.js project
13. Add environment variables (API keys)
14. Deploy!

Environment variables to set:

- GEMINI_API_KEY
- NEXT_PUBLIC_SUPABASE_URL
- NEXT_PUBLIC_SUPABASE_ANON_KEY
- SUPABASE_SERVICE_ROLE_KEY

5.2 Git + GitHub (Version Control)

Repository structure:

```
ai-school/
├── app/                # Next.js app directory
│   ├── api/            # API routes
│   ├── (routes)/       # Page routes
│   └── layout.tsx      # Root layout
├── components/        # React components
├── lib/               # Utilities, helpers
├── public/            # Static assets
├── styles/            # Global styles
└── package.json
```

Branch strategy:

- main - production branch (auto-deploys to Vercel)
- dev - development branch
- feature/* - feature branches

5.3 Domain & DNS

Domain options:

- Use Vercel's free subdomain: your-app.vercel.app
- Or: Buy custom domain (e.g., aischool.ai)

Custom domain setup (if buying):

15. Buy domain (Namecheap, GoDaddy, etc.)
16. Add domain in Vercel dashboard
17. Update DNS records to point to Vercel
18. Automatic HTTPS certificate
19. Live in ~48 hours

6. DEVELOPMENT TOOLS

6.1 Code Editor

Recommended: VS Code or Cursor

VS Code Extensions:

- ES7+ React/Redux/React-Native snippets
- Tailwind CSS IntelliSense
- Prettier - Code formatter
- ESLint
- GitHub Copilot (AI coding assistant)

Cursor (Alternative):

- AI-first code editor (built on VS Code)
- Better AI assistance than Copilot
- Great for vibe coding
- Free tier available

6.2 Package Manager

Options:

- npm (comes with Node.js) - default
- pnpm (faster, saves disk space) - recommended
- yarn (alternative)

Recommendation: Use pnpm
Install: npm install -g pnpm
Usage: pnpm install, pnpm dev, pnpm build

6.3 Code Quality

ESLint (Linting):

- Catches code errors
- Enforces code style
- Comes pre-configured with Next.js
- Run: npm run lint

Prettier (Formatting):

- Auto-formats code
- Consistent style across team
- Install: npm install -D prettier
- Run: npx prettier --write .

.prettierrc config:

```json
{
  "semi": true,
  "singleQuote": true,
  "tabWidth": 2,
  "trailingComma": "es5"
}
```

7. OPTIONAL / NICE-TO-HAVE LIBRARIES

These aren't essential for MVP but can speed up development:

7.1 UI Utilities

clsx / classnames:

- Conditional className composition
- Install: npm install clsx
- Usage: clsx('btn', { 'btn-primary': isPrimary })

lucide-react (Icons):

- Clean, consistent icon set
- Tree-shakeable
- Install: npm install lucide-react
- Usage: import { Mic, Play, Pause } from 'lucide-react'

7.2 Development Utilities

react-hot-toast (Notifications):

- Toast notifications for success/error messages
- Install: npm install react-hot-toast

react-use (Hook collection):

- Useful React hooks
- Install: npm install react-use
- Includes: useLocalStorage, useDebounce, etc.

7.3 Analytics (Optional)

Vercel Analytics:

- Free with Vercel hosting
- Track page views, performance
- Install: npm install @vercel/analytics

PostHog (Product analytics):

- Track user behavior
- Feature flags
- A/B testing
- Free tier available

8. ENVIRONMENT SETUP CHECKLIST

8.1 Prerequisites

- Node.js 18+ installed (download from nodejs.org)
- Git installed
- GitHub account created
- VS Code or Cursor installed

8.2 Service Accounts

Create accounts for:

20. Vercel (vercel.com)
21. Supabase (supabase.com)
22. Google AI Studio (ai.google.dev) - for Gemini API key
23. GitHub (github.com)

8.3 API Keys to Get

24. Gemini API key from ai.google.dev
25. Supabase URL + Anon Key from project settings
26. Supabase Service Role Key (for backend)

8.4 Project Initialization

Step-by-step setup:

1. Create Next.js project:
```
  npx create-next-app@latest ai-school --typescript --tailwind --app
```

2. Install dependencies:
```
  cd ai-school
  pnpm install framer-motion zustand @supabase/supabase-js
  pnpm install react-konva konva
  pnpm install @google/generative-ai
```

3. Install shadcn/ui:
```
  npx shadcn-ui@latest init
  npx shadcn-ui@latest add button card progress dialog
```

4. Create .env.local file:
```
  GEMINI_API_KEY=your_key_here
  NEXT_PUBLIC_SUPABASE_URL=your_url
  NEXT_PUBLIC_SUPABASE_ANON_KEY=your_key
```

5. Initialize Git:

```
git init
git add .
git commit -m "Initial commit"
```

6. Push to GitHub:
   Create repo on GitHub
   ```
   git remote add origin <your-repo-url>
   git push -u origin main
   ```

7. Deploy to Vercel:
   Connect GitHub repo in Vercel dashboard
   Add environment variables
   Deploy!

9. QUICK REFERENCE SUMMARY

Core Stack (Must-Have)

- Next.js 15 + React 18 + TypeScript
- Tailwind CSS + shadcn/ui
- Framer Motion (animations)
- react-konva (whiteboard)
- Zustand (state)
- Next.js API Routes (backend)
- Supabase (database)
- Gemini 3 flash
- Soniox
- Google TTS Neural2
- Web Audio API (voice)
- Vercel (hosting)

Development Commands

```
pnpm install          # Install dependencies
pnpm dev              # Start dev server (localhost:3000)
pnpm build            # Build for production
pnpm start            # Start production server
pnpm lint             # Run ESLint
git push              # Deploy to Vercel (auto)
```

Key File Locations

```
app/page.tsx          # Landing page
app/learn/page.tsx    # Learning interface
app/api/gemini/       # Gemini API routes
components/           # Reusable components
lib/supabase.ts       # Supabase client
lib/store.ts          # Zustand store
.env.local            # Environment variables
```

Important Links

- Next.js docs: nextjs.org/docs
- Tailwind docs: tailwindcss.com/docs
- shadcn/ui: ui.shadcn.com
- Gemini API: ai.google.dev/docs
- Supabase docs: supabase.com/docs
- Vercel docs: vercel.com/docs

Ready to build! In Sha Allah 🚀

** * * END OF TECH STACK DOCUMENT * * **