AI Learning Assistant - Complete Project Documentation

Table of Contents

Project Overview

AI Learning Assistant (Study Buddy) is a full-stack web application that helps students learn more effectively by:

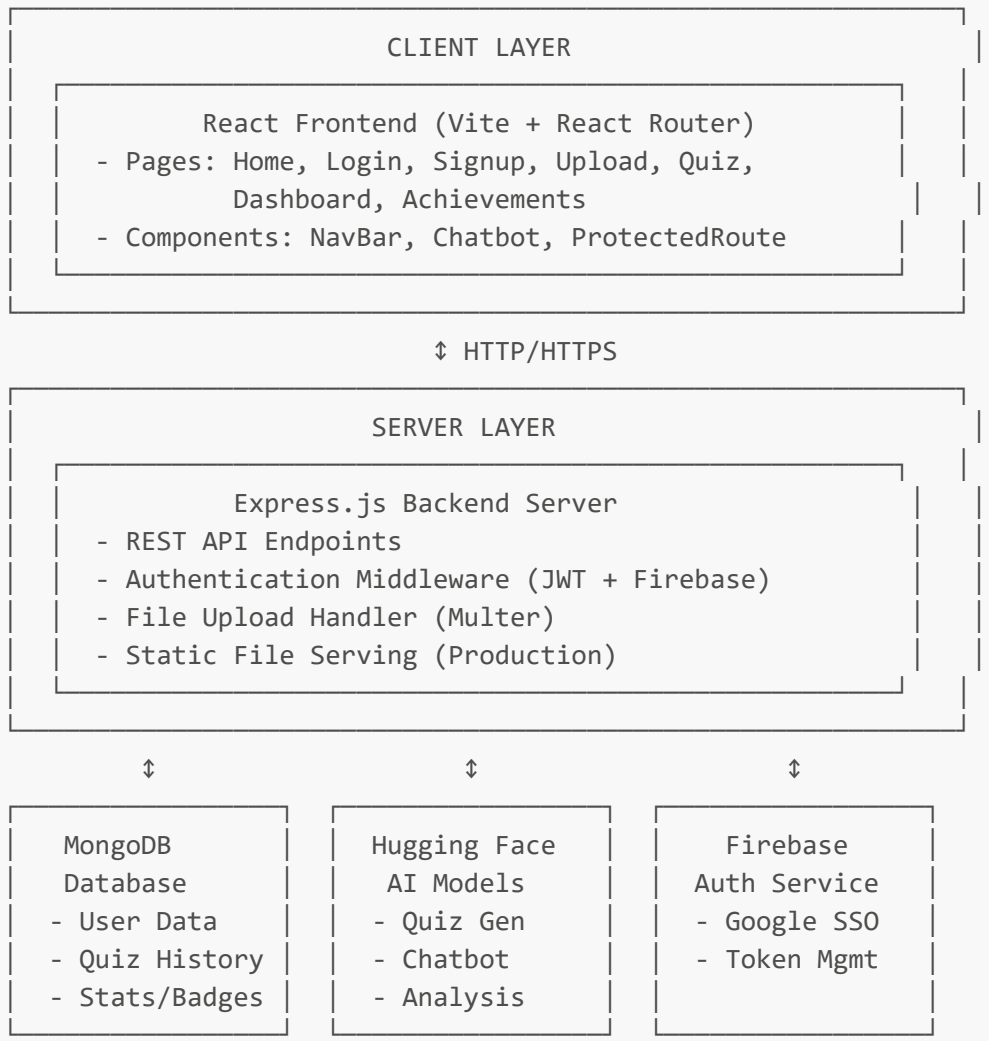
- Generating personalized quizzes from uploaded study materials (PDF/DOCX)
- Providing AI-powered quiz analysis and feedback
- Tracking learning progress with XP, levels, and badges
- Offering an AI chatbot for study assistance
- Maintaining quiz history with detailed analytics

Key Features

- Document upload (PDF, DOCX) with text extraction
- AI-powered quiz generation using Hugging Face models
- Timed and untimed quiz modes
- Comprehensive progress tracking and statistics
- Gamification with XP, levels, badges, and streaks
- AI chatbot for learning assistance
- Dual authentication (JWT + Firebase)
- Responsive design with mobile support

Architecture

High-Level Architecture Diagram



Component Architecture

```
Frontend (React)
├── App.jsx (Router + Chatbot visibility logic)
├── pages/
│   ├── Home.jsx (Landing page)
│   ├── Login.jsx (JWT + Firebase auth)
│   ├── Signup.jsx (User registration)
│   ├── UploadNote.jsx (PDF/DOCX upload)
│   ├── Quiz.jsx (Quiz interface + history)
│   ├── Dashboard.jsx (User stats)
│   └── Achievements.jsx (Badges display)
├── components/
│   ├── NavBar.jsx (Navigation + user profile)
│   ├── Chatbot.jsx (AI assistant)
│   ├── ChatMessage.jsx (Chat UI)
│   └── ProtectedRoute.jsx (Auth guard)
└── utils/
    ├── storeUserData.js
    ├── fetchUserStats.js
    └── updateQuizHistory.js
```

```
Backend (Express)
├── server.js (Main server file)
├── models/
│   └── User.js (MongoDB schema)
└── middleware/
    └── auth.js (JWT + Firebase verification)
```

💻 Technology Stack

Frontend

- **Framework**: React 18
- **Build Tool**: Vite
- **Routing**: React Router v6
- **Styling**: CSS3 (Custom styles)
- **State Management**: React Hooks (useState, useEffect, useMemo)
- **HTTP Client**: Fetch API
- **Authentication**: Firebase Auth SDK

Backend

- **Runtime**: Node.js
- **Framework**: Express.js
- **Database**: MongoDB (Mongoose ODM)
- **Authentication**:
  - JWT (jsonwebtoken)
  - Firebase Admin SDK
- **File Processing**:
  - Multer (file uploads)
  - pdf-parse (PDF extraction)
  - mammoth (DOCX extraction)
- **AI Integration**: Hugging Face Inference API
- **Security**: bcryptjs, CORS

External Services

- **AI Models**: Hugging Face (Qwen/Qwen2.5-7B-Instruct)
- **Database**: MongoDB Atlas
- **Authentication**: Firebase Authentication
- **Email**: SendGrid (optional)

🔄 System Flow

1. User Registration & Login Flow

```
┌──────────┐
│  User    │
└──────┬───┘
       │
       ├─── Option 1: Email/Password Signup
       │    │
       │    ├──> POST /api/signup
       │    │    - Validate input
       │    │    - Hash password (bcrypt)
       │    │    - Create user in MongoDB
       │    │    - Generate JWT token
       │    │    - Return token + user data
       │    │
       │    └──> Store token in localStorage
       │
       └─── Option 2: Google Sign-In
            │
            ├──> Firebase signInWithPopup()
            │    - Get Firebase ID token
            │    - Send to backend
            │
            ├──> POST /api/login (with Firebase token)
            │    - Verify Firebase token
            │    - Find or create user
            │    - Return JWT + user data
            │
            └──> Store token in localStorage
```

2. Quiz Generation Flow

```
┌─────────────────────────────────────────────────────────┐
│ Step 1: Upload Study Material                           │
│  User uploads PDF/DOCX → POST /api/extract-pdf          │
│  - Multer receives file                                 │
│  - Extract text (pdf-parse or mammoth)                  │
│  - Return extracted text                                │
│  - Store in localStorage                                │
└─────────────────────────────────────────────────────────┘
                        ↓
┌─────────────────────────────────────────────────────────┐
│ Step 2: Configure Quiz                                  │
│  User selects:                                          │
│  - Number of questions (10-50)                          │
│  - Difficulty (easy/medium/hard)                        │
│  - Navigate to /quiz page                               │
└─────────────────────────────────────────────────────────┘
                        ↓
┌─────────────────────────────────────────────────────────┐
│ Step 3: Generate Quiz                                   │
│  POST /api/generate-quiz                                │
```

```
│  - Send notes + difficulty + numQuestions                    │
│  - Backend calls Hugging Face API                            │
│  - AI generates mixed MCQ + multiselect questions            │
│  - Parse JSON response                                       │
│  - Return questions array                                    │
│  - Fallback to local generator if AI fails                   │
└──────────────────────────────────────────────────────────────┘
                          ↓
┌──────────────────────────────────────────────────────────────┐
│ Step 4: Choose Quiz Mode                                     │
│  User selects:                                               │
│  - Timed Mode (1 min per question)                           │
│  - Untimed Mode (no time limit)                              │
│  - Set quiz_started flag                                     │
│  - Hide chatbot during quiz                                  │
└──────────────────────────────────────────────────────────────┘
                          ↓
┌──────────────────────────────────────────────────────────────┐
│ Step 5: Take Quiz                                            │
│  - Display questions one by one                              │
│  - Track answers in state                                    │
│  - Show progress bar                                         │
│  - Timer countdown (if timed)                                │
│  - Auto-submit when timer ends                               │
└──────────────────────────────────────────────────────────────┘
                          ↓
┌──────────────────────────────────────────────────────────────┐
│ Step 6: Submit & Analyze                                     │
│  POST /api/submit-quiz                                       │
│  - Calculate score with partial marks                        │
│  - AI generates feedback (Hugging Face)                      │
│  - Return score + analysis + breakdown                       │
└──────────────────────────────────────────────────────────────┘
                          ↓
┌──────────────────────────────────────────────────────────────┐
│ Step 7: Save History & Update Stats                          │
│  POST /api/save-quiz-history                                 │
│  - Calculate XP (base + bonuses)                             │
│  - Update user level                                         │
│  - Update streak                                             │
│  - Check and unlock badges                                   │
│  - Save quiz to history                                      │
│  - Return updated user data                                  │
│  - Show XP/badge notifications                               │
│  - Set quiz_submitted flag                                   │
│  - Show chatbot again                                        │
└──────────────────────────────────────────────────────────────┘
```

3. Chatbot Interaction Flow

```
User types message
      ↓
POST /api/chatbot
      ↓
Include study material context (if available)
      ↓
Call Hugging Face Chat Completions API
      ↓
AI generates response
      ↓
Return response to frontend
      ↓
Display in chat interface
```

Database Schema

User Model (MongoDB)

```
{
  _id: ObjectId,
  name: String,
  email: String (unique, required),
  password: String (hashed),
  firebaseUid: String (optional, for Google sign-in),
  xp: Number (default: 0),
  level: Number (default: 1),

  stats: {
    totalQuizzes: Number,
    totalCorrect: Number,
    totalQuestions: Number,
    totalTimeSpent: Number,
    bestScore: Number,
    averageScore: Number,
    timedQuizzes: Number,
    perfectScores: Number,
    currentStreak: Number,
    longestStreak: Number,
    lastQuizDate: String,
    topicStats: Map {
      "topic_name": {
        count: Number,
        totalScore: Number,
        bestScore: Number,
        avgScore: Number
      }
    }
  },
```

```
  badges: [{
    key: String,
    unlockedAt: Number (timestamp)
  }],

  quizHistory: [{
    id: String,
    name: String,
    topic: String,
    createdAt: Number,
    difficulty: String,
    timed: Boolean,
    elapsedSeconds: Number,
    questions: Array,
    answers: Object,
    score: Number,
    total: Number,
    percent: Number,
    breakdown: Array,
    summary: String
  }],

  createdAt: Date,
  updatedAt: Date
}
```

API Endpoints

Authentication Endpoints

POST /api/signup

Description: Register new user with email/password
Request Body:

```
{
  "name": "John Doe",
  "email": "john@example.com",
  "password": "password123"
}
```

Response:

```
{
  "token": "jwt_token_here",
  "user": {
    "id": "user_id",
    "name": "John Doe",
```

```
    "email": "john@example.com",
    "xp": 0,
    "level": 1
  }
}
```

POST /api/login

Description: Login with email/password or Firebase token
Request Body:

```
{
  "email": "john@example.com",
  "password": "password123"
}
```

Response: Same as signup

Quiz Endpoints

POST /api/generate-quiz

Description: Generate quiz from study notes
Headers: `Authorization: Bearer <token>`
Request Body:

```
{
  "notes": "Study material text...",
  "level": "medium",
  "numQuestions": 20
}
```

Response:

```
{
  "questions": [
    {
      "type": "mcq",
      "q": "What is X?",
      "options": ["A", "B", "C", "D"],
      "correctAnswers": ["B"]
    },
    {
      "type": "multiselect",
      "q": "Select all that apply:",
      "options": ["A", "B", "C", "D"],
```

```
      "correctAnswers": ["A", "C"]
    }
  ]
}
```

POST /api/submit-quiz

Description: Submit quiz answers for grading
Headers: `Authorization: Bearer <token>`
Request Body:

```
{
  "questions": [...],
  "timed": true,
  "timeSpent": 1200
}
```

Response:

```
{
  "score": 85,
  "total": 20,
  "analysis": {
    "score": 85,
    "total": 20,
    "breakdown": [...],
    "summary": "Great job! You showed strong understanding..."
  }
}
```

POST /api/save-quiz-history

Description: Save quiz attempt and update user stats
Headers: `Authorization: Bearer <token>`
Request Body:

```
{
  "attempt": {
    "name": "Math Quiz",
    "topic": "Algebra",
    "score": 17,
    "total": 20,
    "percent": 85,
    ...
  }
}
```

Response:

```
{
  "success": true,
  "history": [...],
  "user": {
    "xp": 250,
    "level": 3,
    "stats": {...},
    "badges": [...]
  },
  "xpGained": 170,
  "bonusXP": 42,
  "newBadges": ["quiz_10", "acc_85"],
  "streakIncreased": true,
  "currentStreak": 5
}
```

GET /api/quiz-history

Description: Get user's quiz history
Headers: `Authorization: Bearer <token>`
Response:

```
{
  "history": [...]
}
```

User Endpoints

GET /api/user-stats

Description: Get user statistics
Headers: `Authorization: Bearer <token>`
Response:

```
{
  "xp": 250,
  "level": 3,
  "stats": {...},
  "badges": [...],
  "name": "John Doe",
  "email": "john@example.com"
}
```

Document Processing

POST /api/extract-pdf

Description: Extract text from PDF/DOCX
Content-Type: `multipart/form-data`
Request: File upload
Response:

```
{
  "text": "Extracted text content...",
  "method": "text"
}
```

Chatbot

POST /api/chatbot

Description: Chat with AI assistant
Request Body:

```
{
  "message": "How do I study better?",
  "context": "Study material context..."
}
```

Response:

```
{
  "response": "Here are proven study techniques..."
}
```

🎨 Frontend Components

Page Components

1. Home.jsx

- Landing page with hero section
- Features overview
- Call-to-action buttons

2. Login.jsx

- Email/password login form
- Google Sign-In button
- Firebase authentication integration
- Redirects to dashboard on success

3. Signup.jsx

- User registration form
- Email/password validation
- Creates user account
- Auto-login after signup

4. UploadNote.jsx

- File upload interface (PDF/DOCX)
- Text paste option
- Document text extraction
- Stores notes in localStorage
- Redirects to quiz page

5. Quiz.jsx

Main Features:

- Quiz configuration (questions, difficulty)
- Quiz mode selection (timed/untimed)
- Question display (MCQ + multiselect)
- Progress tracking
- Timer countdown
- Answer submission
- Score display with AI feedback
- Quiz history with filters/search
- Retake functionality

State Management:

```
- questions: Array of quiz questions
- answers: Object mapping question index to answer
- started: Boolean for quiz state
- submitted: Boolean for submission state
- timedMode: 'timed' | 'untimed'
- timeLeft: Seconds remaining
- score: Final score percentage
- analysis: AI feedback object
- history: Array of past attempts
- filters: search, topic, difficulty, timed, sort
```

6. Dashboard.jsx

- User profile display
- XP and level progress
- Statistics overview
- Recent quiz history

7. Achievements.jsx

- Badge collection display
- Badge unlock conditions
- Progress towards next badges

Reusable Components

NavBar.jsx

- Navigation links
- User profile dropdown
- Logout functionality
- Responsive hamburger menu
- Shows user name, level, XP

Chatbot.jsx

- Floating chat button
- Chat interface
- Message history
- AI response handling
- Hidden during active quiz

ProtectedRoute.jsx

- Authentication guard
- Redirects to login if not authenticated
- Wraps protected pages

🔐 Authentication Flow

JWT Authentication

```
1. User signs up/logs in
2. Backend generates JWT token
3. Token stored in localStorage
4. Token sent in Authorization header for API requests
5. Backend middleware verifies token
6. Request proceeds if valid
```

Firebase Authentication

```
1. User clicks "Sign in with Google"
2. Firebase popup opens
3. User authenticates with Google
4. Firebase returns ID token
5. Frontend sends token to backend
6. Backend verifies with Firebase Admin SDK
7. Backend finds/creates user
8. Returns JWT token
9. Token stored in localStorage
```

Token Refresh

```
- Firebase tokens refreshed automatically
- JWT tokens valid for 7 days
- Frontend refreshes Firebase token before API calls
- Updates localStorage with new token
```

🎮 Gamification System

XP Calculation

```
Base XP = score * 10

Bonuses:
- Difficulty:
  - Easy: +0%
  - Medium: +25%
  - Hard: +50%
- Timed mode: +25%
- Perfect score (100%): +100 XP
- Streak bonus: +10% per day

Total XP = Base XP + Bonuses
```

Level System

```
Level 1: 0 XP
Level 2: 100 XP
Level 3: 200 XP
Level N: (N-1) * 100 XP

Level up when: Current XP >= Level * 100
```

Streak System

```
- Daily streak increments if quiz taken on consecutive days
- Resets if day skipped
- Streak bonus: 10% of base XP per streak day
- Tracks longest streak
```

Badge System

Quiz Milestones:

- first_quiz: Complete 1 quiz
- quiz_5: Complete 5 quizzes
- quiz_10: Complete 10 quizzes
- quiz_25: Complete 25 quizzes
- quiz_50: Complete 50 quizzes
- quiz_100: Complete 100 quizzes

Timed Challenges:

- timed_1: Complete 1 timed quiz
- timed_10: Complete 10 timed quizzes

Accuracy:

- acc_70: Score 70%+
- acc_85: Score 85%+
- acc_95: Score 95%+
- perfect: Get 100% score
- perfect_5: Get 5 perfect scores

Streaks:

- streak_3: 3-day streak
- streak_7: 7-day streak
- streak_30: 30-day streak

Levels:

- level_10: Reach level 10
- level_25: Reach level 25
- level_50: Reach level 50

Badge Rewards: +50 XP per badge unlocked

🚀 Setup Instructions

Prerequisites

- Node.js 18+
- MongoDB Atlas account
- Firebase project
- Hugging Face API key

Backend Setup

1. **Navigate to backend directory**:

```
cd backend
```

2. **Install dependencies**:

```
npm install
```

3. **Create .env file**:

```
PORT=5000
MONGO_URI=your_mongodb_connection_string
JWT_SECRET=your_jwt_secret
HF_API_KEY=your_huggingface_api_key
HF_MODEL=Qwen/Qwen2.5-7B-Instruct
FIREBASE_PROJECT_ID=your_project_id
FIREBASE_CLIENT_EMAIL=your_client_email
FIREBASE_PRIVATE_KEY=your_private_key
FRONTEND_URL=http://localhost:5173
```

4. **Start server**:

```
npm start
```

Frontend Setup

1. **Navigate to frontend directory**:

```
cd frontend
```

2. **Install dependencies**:

```
npm install
```

3. **Create .env file**:

```
VITE_API_URL=http://localhost:5000
VITE_FIREBASE_API_KEY=your_api_key
VITE_FIREBASE_AUTH_DOMAIN=your_auth_domain
VITE_FIREBASE_PROJECT_ID=your_project_id
VITE_FIREBASE_STORAGE_BUCKET=your_storage_bucket
VITE_FIREBASE_MESSAGING_SENDER_ID=your_sender_id
VITE_FIREBASE_APP_ID=your_app_id
```

4. **Start development server**:

```
npm run dev
```

5. **Access application**:

```
http://localhost:5173
```

Deployment Guide

Production Build

1. **Build frontend**:

```
cd frontend
npm run build
```

2. **Set environment variables**:

```
export NODE_ENV=production
```

3. **Start backend**:

```
cd backend
npm start
```

Deployment Platforms

Render / Railway / Heroku

1. Connect GitHub repository
2. Set environment variables
3. Build command: `cd frontend && npm install && npm run build && cd ../backend && npm install`
4. Start command: `cd backend && npm start`

Vercel (Frontend) + Render (Backend)

Frontend (Vercel):

- Deploy frontend directory
- Set VITE_API_URL to backend URL

Backend (Render):

- Deploy backend directory
- Set all environment variables
- Enable static file serving

Environment Variables Checklist

Backend:

- ☑ PORT
- ☑ MONGO_URI
- ☑ JWT_SECRET
- ☑ HF_API_KEY
- ☑ HF_MODEL
- ☑ FIREBASE_PROJECT_ID
- ☑ FIREBASE_CLIENT_EMAIL
- ☑ FIREBASE_PRIVATE_KEY
- ☑ FRONTEND_URL
- ☑ NODE_ENV=production

Frontend:

- ☑ VITE_API_URL
- ☑ VITE_FIREBASE_* (all Firebase config)

🔧 Key Implementation Details

1. Dual Authentication System

```
// JWT for email/password
const token = jwt.sign({ id: user._id }, JWT_SECRET, { expiresIn: '7d' });

// Firebase for Google Sign-In
const decodedToken = await admin.auth().verifyIdToken(firebaseToken);
```

2. Quiz Scoring with Partial Marks

```
// MCQ: 1 or 0
if (userAnswer === correctAnswer) marks = 1;

// Multiselect: Partial credit
const correctSelected = userAnswers.filter(a =>
correctAnswers.includes(a)).length;
const incorrectSelected = userAnswers.filter(a =>
!correctAnswers.includes(a)).length;

if (incorrectSelected > 0) marks = 0;
else if (correctSelected === correctAnswers.length) marks = 1;
else marks = correctSelected / correctAnswers.length;
```

3. AI Integration with Fallback

```
try {
  // Try Hugging Face API
  const response = await
fetch('https://router.huggingface.co/v1/chat/completions', {...});
  const data = await response.json();
  return data.choices[0].message.content;
} catch (error) {
  // Fallback to local generator
  return buildFallbackQuiz(notes);
}
```

4. Chatbot Visibility Control

```
// Hide during active quiz
const quizStarted = localStorage.getItem('quiz_started') === 'true';
const quizSubmitted = localStorage.getItem('quiz_submitted') === 'true';
const showChatbot = !quizStarted || quizSubmitted;
```

5. Static File Serving (Production)

```
if (process.env.NODE_ENV === 'production') {
  app.use(express.static(frontendPath, {
    setHeaders: (res, filePath) => {
      if (filePath.endsWith('.js')) {
        res.setHeader('Content-Type', 'application/javascript');
      }
    }
```

```
  }));
}
```

Performance Optimizations

1. **React useMemo**: Memoize filtered history, stats calculations
2. **Debouncing**: Search input debouncing
3. **Lazy Loading**: Code splitting with React.lazy
4. **MongoDB Indexing**: Email, firebaseUid indexed
5. **API Timeouts**: 30-90s timeouts for AI calls
6. **Fallback Systems**: Local quiz generator, chatbot responses

---

Security Features

1. **Password Hashing**: bcrypt with salt rounds
2. **JWT Tokens**: 7-day expiration
3. **CORS**: Configured for specific origin
4. **Input Validation**: Server-side validation
5. **File Upload Limits**: 10MB max
6. **Environment Variables**: Sensitive data in .env
7. **Firebase Admin SDK**: Secure token verification
8. **Protected Routes**: Authentication middleware

Error Handling

Frontend

```
try {
  const response = await fetch(url, options);
  if (!response.ok) throw new Error('Request failed');
  const data = await response.json();
} catch (error) {
  setError(error.message);
  // Show user-friendly error message
}
```

Backend

```
try {
  // Operation
} catch (error) {
  console.error('Error:', error);
  res.status(500).json({ error: 'Server error' });
}
```

Future Enhancements

1. **Social Features**: Friend system, leaderboards
2. **Study Groups**: Collaborative learning
3. **Spaced Repetition**: Smart review scheduling
4. **Mobile App**: React Native version
5. **Offline Mode**: PWA with service workers
6. **Advanced Analytics**: Learning insights dashboard
7. **Custom Badges**: User-created achievements
8. **Video Integration**: YouTube study material support
9. **Multi-language**: i18n support
10. **Voice Input**: Speech-to-text for chatbot

License

Contributors

- Shashidhar Pawadashetti - Full Stack Developer

Support

For issues or questions, please open an issue on GitHub or contact the development team.

Last Updated: January 2025
Version: 1.0.0
Status: Production Ready ☑