**Technology Stack Overview: Student Management System**

**Frontend Technologies**

**React with TypeScript**

* **What**: Modern JavaScript library with TypeScript for type safety
* **Why**: Provides component-based architecture for building interactive UIs with improved code quality and maintainability through static typing

**React Router**

* **What**: Library for handling navigation in React applications
* **Why**: Enables client-side routing for a smooth, single-page application experience without page reloads

**React Query**

* **What**: Data fetching and state management library
* **Why**: Simplifies API data handling with automatic caching, background updates, and optimistic UI updates

**Tailwind CSS**

* **What**: Utility-first CSS framework
* **Why**: Accelerates UI development with pre-built classes, ensures consistency, and reduces custom CSS needs while maintaining a modern design

**Axios**

* **What**: Promise-based HTTP client
* **Why**: Provides an elegant interface for API requests with features like request/response interception and automatic JSON transformation

**Backend Technologies**

**Express.js (Development)**

* **What**: Lightweight Node.js web framework
* **Why**: Powers our mock server during development with minimal configuration, enabling rapid prototyping

**Flask/Python (Production)**

* **What**: Python web framework
* **Why**: Offers simplicity, flexibility, and extensive libraries for building robust APIs with minimal boilerplate code

**SQLAlchemy ORM**

* **What**: Python SQL toolkit and Object-Relational Mapping library
* **Why**: Abstracts database operations, enabling developers to work with Python objects instead of writing raw SQL

**PostgreSQL (Production) / SQLite (Development)**

* **What**: Relational database management systems
* **Why**: PostgreSQL provides advanced features, reliability, and performance for production, while SQLite offers simplicity for development environments

**API Architecture**

**RESTful API Design**

* **What**: Architectural style for designing networked applications
* **Why**: Provides a standardized, stateless interface that separates client and server concerns, improving scalability and maintainability

**JSON Data Format**

* **What**: Lightweight data interchange format
* **Why**: Human-readable, language-independent format that's easy to parse and generate in multiple programming languages

**API Versioning**

* **What**: Strategy for managing API changes over time
* **Why**: Allows for backward compatibility and smooth transitions when introducing breaking changes

**API Endpoints Structure**

* **Authentication**:

/api/auth/\*

 for user registration, login, and token verification

* **Student Management**:

/api/students/\*

 for CRUD operations on student profiles

* **Study Notes**:

/api/notes/\*

 for managing study materials and progress

* **Wikipedia Integration**:

/api/wikipedia/\*

 for accessing external knowledge resources

* **Analytics**: Specialized endpoints for retrieving performance metrics and insights

**API Documentation**

* **What**: Comprehensive documentation of all API endpoints
* **Why**: Facilitates frontend-backend collaboration and provides a reference for developers

**Deployment & Infrastructure**

**Docker**

* **What**: Containerization platform
* **Why**: Ensures consistent environments across development, testing, and production, eliminating "it works on my machine" problems

**Docker Compose**

* **What**: Tool for defining multi-container Docker applications
* **Why**: Simplifies orchestration of multiple services (frontend, backend, database) with a single configuration file

**Authentication & Security**

**JWT (JSON Web Tokens)**

* **What**: Compact, self-contained tokens for secure information transmission
* **Why**: Enables stateless authentication, reducing database lookups and supporting distributed systems

**CORS (Cross-Origin Resource Sharing)**

* **What**: Security feature that allows or restricts requests from different origins
* **Why**: Protects API from unauthorized cross-origin access while allowing legitimate frontend requests

**Input Validation**

* **What**: Server-side validation of all API inputs
* **Why**: Prevents injection attacks and ensures data integrity

**External API Integration**

**Wikipedia API**

* **What**: Public API for accessing Wikipedia content
* **Why**: Provides educational content to enhance study notes with authoritative information

**Development Tools**

**npm/Node.js**

* **What**: JavaScript runtime and package manager
* **Why**: Manages dependencies and provides build scripts for the frontend application

**Python virtual environments**

* **What**: Isolated Python environments
* **Why**: Prevents dependency conflicts between projects and ensures reproducible environments

This technology stack was carefully selected to balance development speed, application performance, and maintainability while providing a modern user experience and robust API architecture.