פרויקט ORT: מערכת Q&A חכמה לכיתה

מערכת שאלות ותשובות מתקדמת המשלבת בינה מלאכותית לשיפור חווית הלמידה. המערכת מיועדת למורים וסטודנטים, ומציעה חיפוש סמנטי חכם וניתוח תשובות אוטומטי.

תכונות מרכזיות

- ניהול שאלות כיתתיות ותשובות סטודנטים בצורה יעילה ומאורגנת
- תשובות תלמידים: שליחה וניהול של תשובות תלמידים
- בקרת גישה: אימות מורים מבוסס קוד גישה
- חיפוש חכם: חיפוש סמנטי באמצעות שאילתות שפה טבעית
- סיכום בינה מלאכותית: ניתוח תשובות תלמידים באמצעות הוראות מותאמות אישית

ארכיטקטורה

תיאור התרשים:

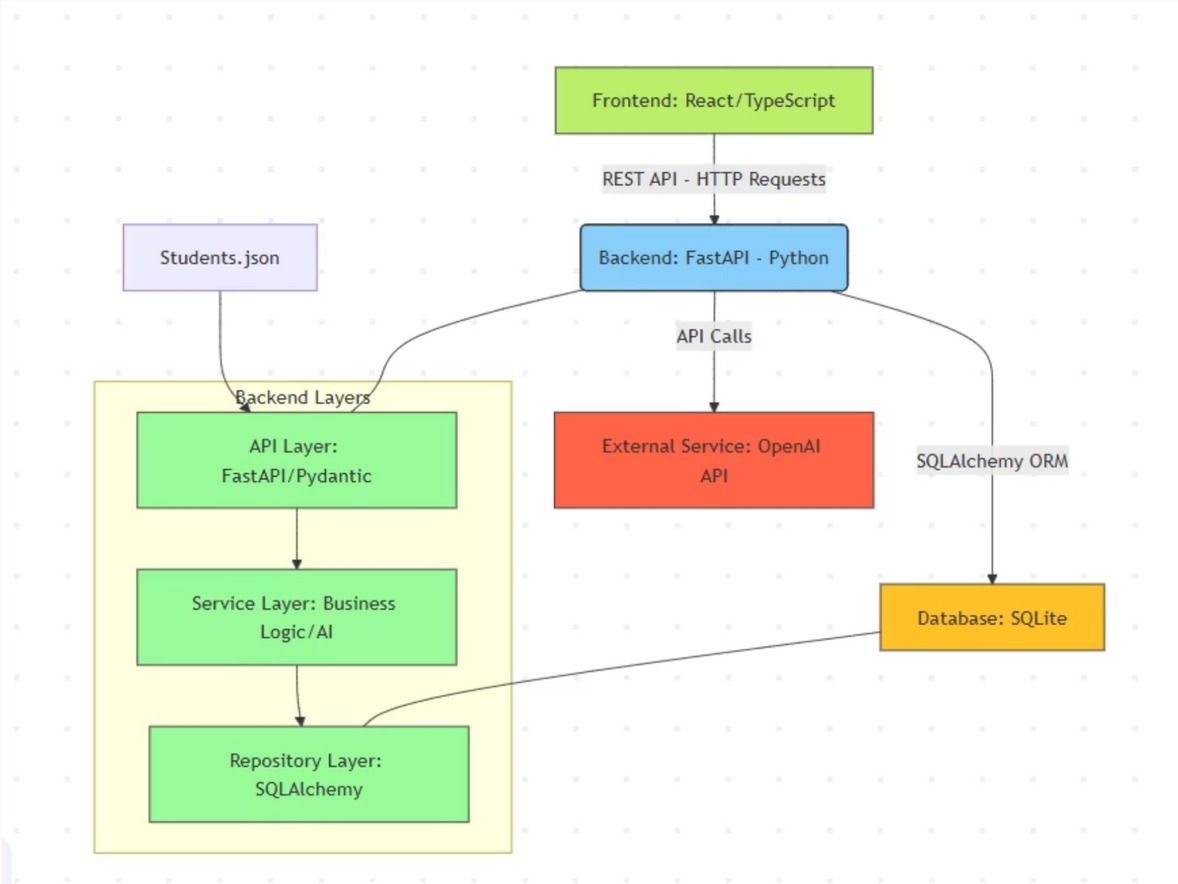
צד לקוח: אפליקציית React/TypeScript שמציגה את הממשק למורה ולתלמיד.

תקשורת (REST API): ה-Frontend שולח בקשות HTTP ל-Backend.

צד שרת: יישום FastAPI ב-Python המטפל בלוגיקה העסקית.

שכבות פנימיות (Layers): מציג את חלוקת האחריות הפנימית של ה-Backend ל-API, Service ו-Repository.

שירות חיצוני: שירות OpenAI API לביצוע Smart Search ו-AI Summarization.



ספריות ושירותים

Frontend - לקוח

React 18

ספריית UI: ממשק משתמש מודרני מבוסס קומפוננטות פונקציונליות ו-Hooks

TypeScript

בטיחות טיפוסים: קוד בטוח יותר, הפחתת שגיאות ושיפור חווית פיתוח

Tailwind CSS

CSS Utility-First: פיתוח מהיר של עיצוב רספונסיבי

Vite

Build Tool: כלי בנייה מהיר עם HMR

Backend - שרת

FastAPI

שרת API: פיתוח מהיר של REST API עם ביצועים גבוהים (ASGI/Uvicorn)

Pydantic

ולידציה: בדיקת תקינות נתונים נכנסים ויוצאים, כולל נתוני AI

SQLAlchemy

ORM: ניהול אינטראקציה עם SQLite בצורה אובייקטית

Uvicorn

שרת ASGI: שרת אסינכרוני עם תמיכה

pytest

בדיקה: סיפרייה לצורכי טסטים לקוד

Openai

ספרייה לעבודה נוחה עם המודל chatGPT

כלים נילווים לתמיכה ופיתוח

הכלים הבאים תומכים ומבטיחים פיתוח, בדיקה ופריסה חלקה של הפרויקט, תוך שמירה על איכות קוד גבוהה ואוטומציה מלאה.

Cursor

פיתוח תוכנה: כלי לפיתוח תוכנה IDE המשלב בתוכו יכולות פיתוח AI בעזרת צ'ט שיודע לקרוא context שך הקוד. יצירת קובץ cursor rule ליצרת חוקים עבור LLM לפיתוח אחיד וחלק יותר.

DBeaver

ניהול נתונים: כלי GUI אינטואיטיבי לבדיקה, הרצת שאילתות וניפוי שגיאות של SQLite

Docker

קונטיינריזציה: סביבת פיתוח אחידה עם docker-compose.yml המאפשרת הרצה מיידית של Backend ו-Frontend יחד

Postman

בדיקות תקשורת: כלי המיועד לבדיקות ה- API של השרת.

GitHub Actions

CI/CD: הפעלת בדיקות אוטומטיות על כל Commit ו-Pull Request, עם בדיקות מטריצה לגרסאות Python ו-Node.js

Gemini

ייעוץ והכוונה: LLM לצורכי ייעוץ ומענה לשאלות במהלך הפיתוח.

Vitest

בדיקות לקוח: Framework מהיר לבדיקות קומפוננטות React, Hooks וזרימות משתמש (User Workflows).

pytest

בדיקות שרת: Framework לבדיקות יחידה (Unit) ואינטגרציה (Integration) של שירותי השאלות, התשובות וה-API.

מבנה Backend בארכיטקטורה שכבתית

השרת בנוי בארכיטקטורה שכבתית המבטיחה הפרדת דאגות ובדיקותיות גבוהה:

API Layer

ניתוב קריאות - POST /api/v1/questions/open

טיפול בתקלות בעזרת בעזרת קובץ error_handler

Service Layer

לוגיקה עסקית - QuestionService, AnswerService

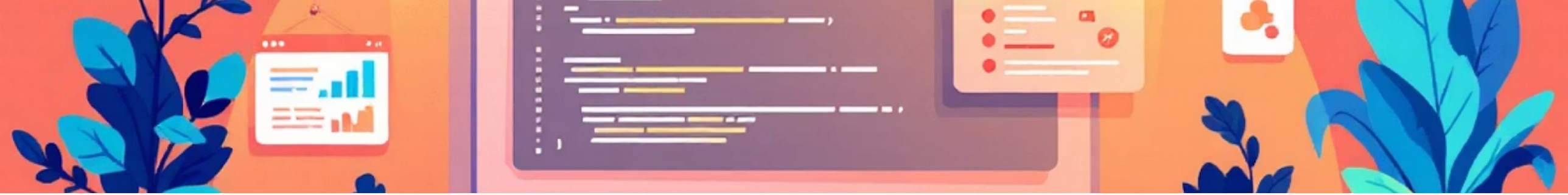
Repository Layer

גישה לבסיס נתונים - QuestionRepository

Data Layer

מודלים - Question Model, Answer Model

```
backend/
├── app/                          # Main application code
│   ├── __init__.py              # Package initialization
│   ├── main.py                  # FastAPI application
entry point
│   ├── api/                     # API layer
│   │   ├── api.py               # Main API router
configuration
│   │   └── endpoints/           # API endpoint modules
│   ├── config/                  # Configuration modules
│   ├── database/                # Database layer
│   │   ├── config.py            # Database configuration
and connection
│   │   ├── models/              # SQLAlchemy ORM models
│   │   └── repositories/        # Data access layer
│   ├── models/                  # Pydantic models for API
│   ├── services/                # Business logic layer
│   └── utils/                   # Utility functions
├── tests/                       # Test suite
├── requirements.txt             # Python dependencies
├── pytest.ini                  # pytest configuration
├── run_tests.py                # Test runner script
├── set_db_path.py              # Database path
configuration
├── database_schema.sql         # Database schema
definition
├── Dockerfile                  # Docker configuration
├── .dockerignore               # Docker ignore file
├── .gitignore                  # Git ignore file
├── README.md                   # Backend documentation
```



מבנה Frontend

מבנה קבצים מודולרי המבטיח קריאות, ארגון וקלות תחזוקה. כל רכיב במערכת ממוקם בתיקייה ייעודית עם הפרדה ברורה בין שרת ולקוח.

```
ort-frontend/src/
├── components/            # Reusable UI
components
├── pages/                 # Page components
├── hooks/                 # Custom React hooks
├── types/                 # TypeScript type
definitions
├── theme.css              # Custom Tailwind CSS
theme
├── App.tsx                # Main application
component
└── main.tsx               # Application entry
point
```

מבנה התיקיות הראשי

פיתוח

שימוש ב-**TypeScript** ו-**Vite** לשמירה על Type Safety ובניית פרויקט מהירה.

טכנולוגיות

מותאמים אישית (Custom Hooks) עם Hooks React 18 ואחזור נתונים מה State לניהול API.

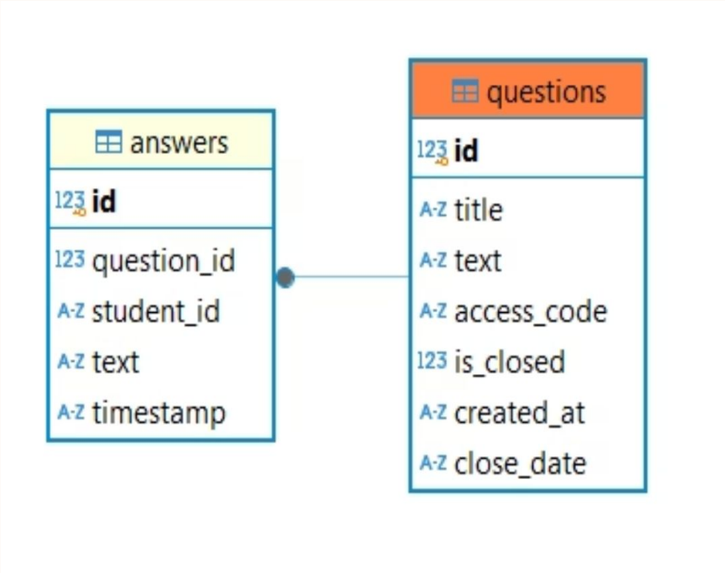
עיצוב ו-Styling

שימוש ב-**Tailwind CSS** ליצירת עיצוב Responsive (Mobile-first) ומהיר, עם דגש על **נגישות (Accessibility)**

בסיס הנתונים - SQLite

בסיס נתונים קל משקל מבוסס דיסקת, פשוט להטמעה, ללא צורך בהגדרת שרת נתונים נפרד, אידיאלי לסביבת פיתוח. עבור בדיקות, נעשה שימוש ב-**in-memory SQLite** מבודד עבור כל הרצת בדיקה, מה שמבטיח נתונים נקיים ועצמאיות בין הבדיקות.

גרף ERD המציין קשר של אחד לרבים בין התשובות לשאלות



גישה לנתונים (Data Access) – שכבת ה-ORM

- **Object-Relational Mapper (ORM)** - SQLAlchemy: ספריית ה-ORM שנבחרה. מאפשרת עבודה עם בסיס הנתונים באמצעות אובייקטי Python
- **Data Models** - מוגדרים מודלים כגון **Question Model** ו-**Answer Model**.
- **Repository Layer** - הגישה לבסיס הנתונים נעשית דרך **שכבת ה-Repository**, שמטרתה להפריד את לוגיקת הגישה לנתונים מהלוגיקה של הסרוויס(Service Layer).

מערכת AI לאנליזה וחיפוש

פונקציונליות מרכזית	/ai/summarize (סיכום תשובות)	/ai/smart-search (חיפוש חכם)
מטרה	יצירת סיכומים מקיפים לתשובות סטודנטים.	ביצוע חיפוש סמנטי (שפה טבעית) של שאלות.
קלט עיקרי	שאלת הקשר + רשימת תשובות + הוראות סיכום.	שאילתת חיפוש + רשימת שאלות זמינות.
פלט	טקסט סיכום (שפה טבעית).	רשימת מזהי שאלות (IDs) תואמים.
מצב תגובה	**Text Response** (שפה טבעית)	**JSON Response** (נתונים מובנים)

ה API: Chat Completions API של OpenAI (מודל ברירת מחדל: `gpt-3.5-turbo`).

אינטגרציה: בקשות HTTP ישירות באמצעות `requests` (במקום SDK) לשליטה מרבית.

אבטחה וחוסן:

- ממשתני סביבה `OPENAI_API_KEY` שימוש ב-.
- הנדסת פרומפטים מותאמת לכל מקרה שימוש.

כלי AI בפיתוח – ניתוח, אינטגרציה ותובנות מרכזיות

כלי AI	שימוש/תפקיד עיקרי	אתגרים מרכזיים	תובנות ושיטות עבודה מומלצות
1. Cursor AI (כלי פיתוח עיקרי)	**הבנה וניתוח קוד:** ניתוח בסיס קוד קיים, הבנת זרימה וזיהוי דפוסים ארכיטקטוניים.	**מגבלות חלון ההקשר:** צורך בבחירה סלקטיבית של קבצים לקריאה.	**MCP (Model Context Protocol):** שימוש ב-**Postman** לשיפור הדיוק על ידי **Context7**-וב הרחבת הקונטקסט (API, תיעוד).
	שימוש מתקדם: אכיפת סגנון ועקביות קוד.	**דיוק ההיסק:** לעיתים נדרשה קריאה חוזרת להבנה מלאה.	חיוני לשמירה על **Cursor Rules:** סטנדרטים ארכיטקטוניים וסגנון קוד.
2. OpenAI API (אינטגרציית Backend)	**שירותי AI מובנים:** שירות סיכום (לתשובות סטודנטים) ושירות חיפוש סמנטי.	**הנדסת פרומפטים:** דרישה לניסוח קפדני של System Prompts לקבלת עקביות.	**JSON Response Format:** שימוש ב-`response_format: {"type": "json_object"}` הבטיח אמינות גבוהה בפלט.
3. Gemini (ייעוץ וניתוח)	**ראייה ארכיטקטונית:** ניתוח מבנה מערכת כללי, אימות דפוסי עיצוב (Best Practices).	**מעבר הקשר:** קושי בשמירה על עקביות ההבנה במעבר בין כלי AI שונים.	**חוזקות משלימות:** כל כלי מצטיין בתחום אחר (קוד, אינטגרציה, ארכיטקטורה).
			[illegible] ל שירות להכין תשובה