



Team Name: ConceptCatch

Team Members:

Abdallah Adel Shabaan Abdallah

Shahd Mohamed Saber Ammar

Yasmeen Sameh Mohamed Ahmed

Date: October 12, 2025

Table of Contents

Contents

1	Problem Description and Market / User Analysis	2
2	Detailed Solution Description	2
2.1	Overview	2
2.2	Architecture	3
2.3	Technology Stack	3
2.4	Main Functionalities	3
2.4.1	1. AI-Powered Feedback for Students	3
2.4.2	2. Quiz and Problem Set Creation	4
2.4.3	3. Mistake Analysis and Learning Path Optimization	5
3	In-depth Analysis of Generative AI Usage	5
3.1	AI Usage Across SDLC Stages	5
3.2	Effectiveness and Benefits	6
3.3	Challenges and Limitations	6
4	Testing and Quality Assurance	9
4.1	Unit and Integration Tests	9
4.2	AI-assisted Testing	9
4.3	Performance and Load Testing	9
4.4	Acceptance Criteria	9
5	Challenges Faced, Solutions Implemented, and Key Lessons Learned	9
5.1	Major Challenges	9
5.2	Solutions Implemented	10
5.3	Key Lessons Learned	10
6	Future Enhancements and Potential Impact	10
6.1	Planned Features	10
6.2	Potential Impact	11

1. Problem Description and Market / User Analysis

Students often make recurring mistakes in quizzes, but instructors lack scalable methods to analyze *why* those mistakes occur. Traditional assessment systems typically mark answers as simply “wrong,” providing no insight into whether the error stems from conceptual misunderstanding, procedural confusion, or a simple oversight. This limits both students’ ability to learn from feedback and teachers’ capacity to identify broader learning trends.

Target Users

- **Students:** Seeking detailed feedback and personalized improvement paths.
- **Instructors:** Wanting aggregated analytics to understand common learning gaps across classes.

Market Gap

While many Learning Management Systems (LMS) such as Moodle and Google Classroom offer automated grading, few leverage **Generative AI** to perform deep mistake analysis or generate personalized insights that transform errors into opportunities for learning.

2. Detailed Solution Description

2.1. Overview

ConceptCatch is an AI-powered web application designed to transform student mistakes into personalized learning opportunities. It uses Generative AI to evaluate student answers, classify mistake types, explain errors in clear language, and reteach misunderstood concepts through adaptive examples and practice. The platform allows students and teachers to create custom or AI-generated quizzes, receive instant feedback, and view detailed analytics through an interactive dashboard. By combining automated assessment, personalized reteaching, and performance tracking, ConceptCatch enhances learning efficiency and helps educators identify conceptual gaps at both individual and class levels.

2.2. Architecture

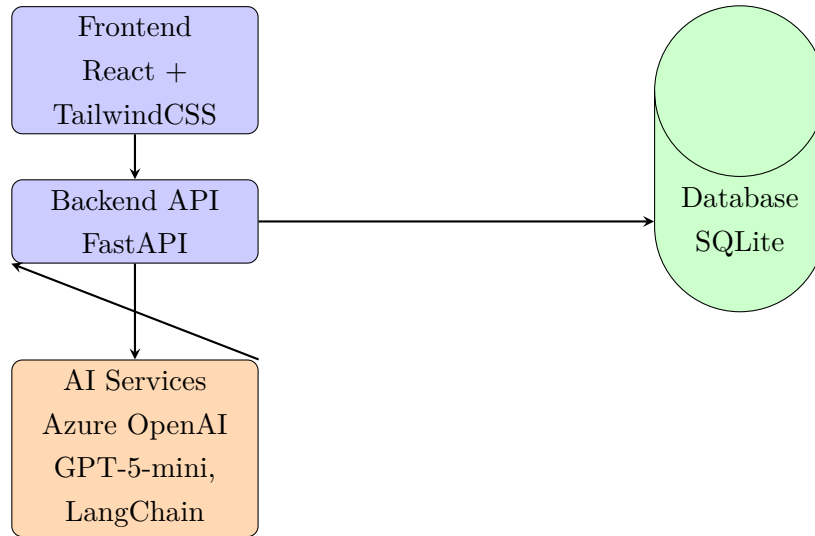


Figure 1: ConceptCatch System Architecture

2.3. Technology Stack

Layer	Technology	Purpose
Frontend	React, TailwindCSS	Build responsive user interfaces, dashboards, and interactive views.
Backend	FastAPI	Serve REST APIs, handle asynchronous operations, and manage business logic.
Database	SQLite	Store structured data and perform analytics queries efficiently.
AI	Azure OpenAI GPT-5-mini, langchain_openai	Perform classification, embeddings, summarization, processing, and text generation tasks.

Table 1: Technology stack used in ConceptCatch.

2.4. Main Functionalities

The ConceptCatch platform consists of three core functional modules designed to create an intelligent, feedback-oriented learning environment: (1) AI-Powered Feedback for Students, (2) Quiz and Problem Set Generation, and (3) Mistake Analysis and Learning Path Optimization.

2.4.1 1. AI-Powered Feedback for Students

Core Capabilities:

- Detect correctness of student responses (right, wrong, partially correct).
- Classify mistake type (conceptual, procedural, arithmetic, misread, reasoning, communication, strategic, careless, misinterpretation, or formatting).
- Explain why an answer is incorrect using clear, student-friendly language.
- Reteach the concept through step-by-step breakdowns or analogies.
- Generate similar practice problems tailored to the same concept.

Input Sources:

- Manual question-answer (QA) entry directly on the platform.
- Ability to upload a reference file that the system uses to verify and evaluate the correctness of student answers.

User Flow (Student):

1. Submit question and answer (or import previous attempts).
2. AI checks correctness and classifies the error.
3. Explanation and reteaching are provided instantly.
4. Optional similar practice questions are generated.
5. Mistake is logged for tracking and analytics.

2.4.2 2. Quiz and Problem Set Creation**Core Capabilities:**

- Topic and difficulty selection.
- Multiple question types supported (MCQ, short answer, essay, and true/false.).
- AI-generated quizzes and teacher-edited quiz versions.
- Automatic generation of detailed explanations.
- Adaptive quizzes based on the student's past mistakes for targeted practice.

User Flow (Student):

1. Select **Practice Mode** — choose a topic and difficulty level, select the “Based on My Mistakes” mode, or upload a document from which the system can automatically generate a quiz.
2. Pick question types.
3. Choose the number of questions.

4. AI generates a quiz with answers and explanations.
5. Take the quiz and receive instant feedback.
6. Retry with new or similar questions generated by the system.

User Flow (Teacher):

1. Select “Create Quiz” mode — choose topics, number of questions, and difficulty level.
2. Pick question types.
3. AI generates draft questions — teacher reviews and edits as needed.
4. Save or publish the quiz.
5. Share the quiz via exporting it to PDF/DOCX format.

2.4.3 3. Mistake Analysis and Learning Path Optimization

Core Capabilities:

- Label mistake types and aggregate them over time.
- Identify recurring weakness points or misunderstood concepts.
- Generate targeted quizzes from a student’s mistake history.
- Provide a teacher dashboard with detailed, exportable performance reports.

User Flow (Student):

1. Mistakes are automatically logged after each quiz attempt.
2. Key weakness points are highlighted in the analytics dashboard.
3. AI recommends targeted practice quizzes based on mistake clusters.

3. In-depth Analysis of Generative AI Usage

This section provides a critical reflection on the effectiveness, challenges, and benefits of using **Generative AI** throughout different stages of the Software Development Life Cycle (SDLC). It also analyzes how AI impacted development speed, code quality, learning, and problem-solving within the **ConceptCatch** project.

3.1. AI Usage Across SDLC Stages

- **Requirements and Brainstorming:** Generative AI was used as a brainstorming partner to refine the project scope, identify user needs, and propose features aligned with the market gap.
- **System Design:** AI provided assistance in suggesting system architecture, database relationships, and integration pathways between FastAPI, LangChain, and Azure OpenAI.

- **Implementation:** Generative AI supported the team in writing code snippets, generating API routes, and structuring prompt templates for mistake classification and feedback.
- **Testing and Debugging:** AI assisted in debugging the code and creating test cases to verify its functionality
- **Documentation and Reporting:** Generative AI was leveraged to summarize development stages, refine technical writing, and assist in report drafting.

3.2. Effectiveness and Benefits

Generative AI improved **development speed** by accelerating prototyping and reducing boilerplate coding. It enhanced **code quality** through structured suggestions and aided **learning efficiency** by clarifying complex tools like LangChain and Azure OpenAI. The collaborative use of AI also stimulated creativity in feature design and problem-solving.

3.3. Challenges and Limitations

- **Prompt Sensitivity:** Output quality varied depending on prompt clarity and structure.
- **Human Oversight:** AI occasionally produced incomplete or ambiguous results, reinforcing the need for manual validation.

Example 1: Debugging LangChain Memory Error

Usage: Debugging LangChain memory error

Our prompt:

I'm using LangChain memory to remember the question, the student's previous answer, and the AI feedback when handling a follow-up question. I want the LLM to be aware of the original question to generate similar practice questions if requested.

I encountered the following error:

```
ValueError: One input key expected, got ['followup_format_instructions', 'prev_question']
```

This happens when I include `prev_question` in the LLM chain.

Code snippet:

```
1 followup_prompt = ChatPromptTemplate.from_messages([
2     ("system",
3     """You are a helpful tutor continuing a conversation with a
4     student.
5     Use the previous discussion (question, student answer, tutor
6     feedback,
7     and any provided resource) as authoritative context.
8     When the student requests practice questions, generate 3 similar
9     practice questions based only on the previous question: {
10    prev_question}
```

```
8         by default, unless the student specifies a different number.
9         Answer the student's follow-up question clearly and kindly.""
10    ),
11    ("user", "Followup_Question: {followup_question}")
12 ])
13
14 chain = LLMChain(llm=llm, prompt=followup_prompt, memory=memory)
15 result = chain.invoke({
16     "followup_question": followup_question,
17     "prev_question": prev_question
18 })
```

ChatGPT response:

ConversationBufferMemory expects exactly one input key for user input. You passed a dictionary with two keys:

```
1 {
2     "followup_question": followup_question,
3     "prev_question": prev_question
4 }
```

Solution: Instead of keeping {prev_question} as a placeholder, use an f-string when creating the prompt:

```
1 followup_prompt = ChatPromptTemplate.from_messages([
2     ("system", f"""
3     You are a helpful tutor continuing a conversation with a student.
4     The previous question is: {prev_question}
5     ...
6     """),
7     ("user", "Followup_Question: {followup_question}")
8 ])
```

Outcome: The fix resolved the error and allowed the model to maintain conversational context. This improved follow-up question generation and ensured more coherent multi-turn interactions.

Example 2: Debugging Asynchronous API Initialization and Environment Configuration in FastAPI

Usage: Fixing a runtime issue that caused the FastAPI backend to fail during startup when initializing LangChain and Azure OpenAI clients. The problem appeared after adding asynchronous route handlers and modularizing the project into separate routers.

Our prompt:

After modularizing our FastAPI app into multiple routers, the Azure OpenAI connection stopped working. We are using LangChain's `AzureOpenAI` class for model initialization. When we start the server, we get:


```
openai.error.AuthenticationError: No API key provided.
```

Even though the .env file exists and contains:

```
1 AZURE_OPENAI_API_KEY=xxxx
2 AZURE_OPENAI_ENDPOINT=https://conceptcatch-openai.openai.azure.com
```

We load environment variables in config.py as follows:

```
1 from dotenv import load_dotenv
2 import os
3 load_dotenv()
4
5 api_key = os.getenv("AZURE_OPENAI_API_KEY")
6 endpoint = os.getenv("AZURE_OPENAI_ENDPOINT")
```

This worked before we made the routes async. Could async startup be interfering with environment loading or dependency injection?

GPT-5 Mini response:

The AI identified that `load_dotenv()` was being called *after* the FastAPI event loop started, which caused asynchronous tasks depending on environment variables to initialize before they were properly loaded.

It suggested several fixes:

1. Move the environment loading logic into the FastAPI `lifespan` event or the main entry file (`main.py`), before importing routers.
2. Use lazy initialization for the Azure OpenAI client to ensure it is created only when first needed:

```
1 from functools import lru_cache
2 @lru_cache
3 def get_llm():
4     from langchain_openai import AzureOpenAI
5     return AzureOpenAI(deployment_name="gpt-5-mini")
```

3. Add an asynchronous startup event to verify configuration before serving requests:

```
1 @app.on_event("startup")
2 async def verify_env():
3     assert os.getenv("AZURE_OPENAI_API_KEY") is not None, "Missing API key"
```

Outcome: After refactoring initialization order and dependencies, the backend launched successfully and maintained stable asynchronous connections with Azure OpenAI. This eliminated race conditions and reduced cold-start latency in deployment.

4. Testing and Quality Assurance

4.1. Unit and Integration Tests

Formal automated unit and integration tests were not implemented due to the project's prototypical scope.

4.2. AI-assisted Testing

Artificial Intelligence tools played an indirect yet valuable role in the testing and debugging phases. ChatGPT was used to assist in identifying potential logic errors, improving error-handling mechanisms, and suggesting optimized debugging statements in specific sections of the backend code. Additionally, AI assistance helped in generating synthetic inputs and edge-case scenarios to verify the robustness of the text classification and summarization pipelines.

4.3. Performance and Load Testing

Although large-scale load testing was not within the project's scope, general performance was monitored during development. API response times were tested under typical user requests, ensuring that endpoints responded efficiently with low latency.

4.4. Acceptance Criteria

The system was considered acceptable when the following conditions were achieved:

- FastAPI endpoints consistently produced correct and timely responses.
- Database operations (insertion, retrieval, and updates) executed without error.
- The AI classification and summarization results were contextually relevant and coherent.
- The overall user interaction between frontend and backend components was stable and error-free.

5. Challenges Faced, Solutions Implemented, and Key Lessons Learned

5.1. Major Challenges

Throughout the development of ConceptCatch, several significant challenges were encountered:

- **Limited familiarity with Generative AI technologies:** At the start of the project, the team had limited prior experience with generative AI and modern LLM-based workflows. This made it difficult to decide which technologies and frameworks to use for the application pipeline.
- **Backend integration complexity:** Integrating various components—such as FastAPI endpoints, the database layer, and AI services—proved to be highly challenging. The backend required managing asynchronous operations, structured API routes, and consistent data exchange between the frontend and AI models.

- **Access to Azure OpenAI services:** Initially, the team was unaware of the Azure OpenAI access model. Since the OpenAI API is paid, finding a feasible solution within student budget constraints required exploring the Azure student credits program and configuring regional availability.

5.2. Solutions Implemented

To address these challenges, the following strategies were applied:

- The team completed a 30-hour intensive course in Generative AI and Large Language Models to build foundational understanding. This was followed by incremental experimentation—starting with small-scale tasks such as question generation and gradually evolving toward full pipeline integration.
- Backend issues were resolved through extensive debugging and iterative development. Online documentation, error tracing, and AI-assisted debugging were used to identify and fix integration issues. This hands-on learning process significantly improved backend development skills.
- Azure OpenAI was integrated after discovering the availability of free student credits, which provided access to GPT-4 and GPT-5-mini models. This enabled the deployment of high-quality language services without incurring additional costs.

5.3. Key Lessons Learned

The development process offered valuable lessons that extend beyond technical implementation:

- Continuous learning and experimentation are essential when working with rapidly evolving AI technologies.
- Backend development requires both structural planning and persistent debugging to ensure reliable integration.
- Leveraging available academic resources and cloud credits can significantly reduce project costs and enable access to enterprise-level tools.
- Effective collaboration and task division within the team enhance productivity, even when all members are learning new technologies.

6. Future Enhancements and Potential Impact

6.1. Planned Features

- Extend mistake tracking to include errors identified in manually entered questions, ensuring all learning interactions are analyzed consistently.
- Enable batch uploading of multiple questions for automated evaluation, improving efficiency and usability for instructors handling large question sets.

- Integrate ConceptCatch with popular Learning Management Systems (LMS) such as Google Classroom and Moodle through a browser extension, allowing users to access AI-powered feedback directly within their existing platforms.
- Provide enhanced teacher analytics dashboards to review both individual and class-wide performance trends, supporting data-driven teaching decisions.
- Slides or PPT generator as an important feature for teacher and students can be added.
- Summarization of uploaded document from students and using them to generate study plans.
- Providing visuals and voice explanation for questions by using text-to-voice models and integrating them with manim which act as a board and AI teaching on it.

6.2. Potential Impact

These enhancements will significantly improve ConceptCatch's accessibility, efficiency, and analytical capabilities. By connecting with established LMS platforms, the system can reach a wider audience while offering seamless user experiences. Expanded analytics and batch processing will empower educators with deeper insights into learning patterns and help students receive continuous, context-aware feedback across all forms of assessments.

References

The following references represent the main technologies, frameworks, and APIs used in the development of the ConceptCatch platform.

1. React Documentation. Available at: <https://react.dev/reference/react>
2. FastAPI Documentation – Virtual Environments. Available at: <https://fastapi.tiangolo.com/virtual-environments/>
3. SQLite Documentation. Available at: <https://www.sqlite.org/docs.html>
4. LangChain API Reference – AzureChatOpenAI. Available at: https://python.langchain.com/api_reference/openai/chat_models/langchain_openai.chat_models.azure.AzureChatOpenAI.html#langchain_openai.chat_models.azure.AzureChatOpenAI
5. LangChain API Reference – AzureOpenAIEmbeddings. Available at: https://python.langchain.com/api_reference/openai/embeddings/langchain_openai.embeddings.azure.AzureOpenAIEmbeddings.html#langchain_openai.embeddings.azure.AzureOpenAIEmbeddings