### ThinkEdge - Technical Architecture & Source Code Overview

* **Document Version:** 1.1
* **Date:** August 7, 2025
* **Status:** Final Pre-GitHub Draft

### 1. High-Level System Architecture

The application is built on a modern, scalable, and language-agnostic three-tier architecture, with the Gemini API acting as the intelligent fourth tier. This modular design ensures that each part of the system can be developed, updated, and scaled independently.

+---------------------+ +---------------------+ +---------------------+  
| | | | | |  
| Android & iOS |<---->| Backend Server |<---->| Database |  
| (Kotlin/SwiftUI) | | (Python/FastAPI) | | (Firestore) |  
| | | | | |  
+---------------------+ +----------+----------+ +---------------------+  
 |  
 | (Secure API Calls)  
 |  
 +--------v--------+  
 | |  
 | Gemini API |  
 | |  
 +-----------------+

### 2. Component 1: The Mobile Application (Android)

This is the primary user-facing component. It is built with a focus on performance, offline capability, and a clean user experience.

* **Language:** Kotlin
* **UI Toolkit:** Jetpack Compose
* **Architecture:** MVVM (Model-View-ViewModel)

#### Key File Structure & Explanation:

/app  
└── /src  
 └── /main  
 ├── /java/com/thinkedge  
 │ ├── /data  
 │ │ ├── /model // Data classes (User, Lesson, etc.)  
 │ │ ├── /repository // Manages data sources (network/local)  
 │ │ └── /network // Retrofit API service definitions  
 │ ├── /di // Hilt dependency injection modules  
 │ ├── /ui  
 │ │ ├── /screens // Composable functions for each screen (Dashboard, Lesson)  
 │ │ ├── /viewmodel // ViewModels for each screen  
 │ │ └── /theme // App theme and colors  
 │ └── MainActivity.kt // The main entry point of the app  
 └── /res // Android resources (icons, strings)

#### Source Code Snippet: Dashboard ViewModel (Simplified)

This snippet shows how the ViewModel for the main dashboard would be structured to manage the user's state and handle events.

// /ui/viewmodel/DashboardViewModel.kt  
  
@HiltViewModel  
class DashboardViewModel @Inject constructor(  
 private val userRepository: UserRepository,  
 private val lessonRepository: LessonRepository  
) : ViewModel() {  
  
 private val \_uiState = MutableStateFlow<DashboardUiState>(DashboardUiState.Loading)  
 val uiState: StateFlow<DashboardUiState> = \_uiState  
  
 init {  
 viewModelScope.launch {  
 // Combine multiple data sources into a single UI state  
 combine(  
 userRepository.getCurrentUser(),  
 lessonRepository.getCurrentProgress()  
 ) { user, progress ->  
 DashboardUiState.Success(user = user, progress = progress)  
 }.catch { exception ->  
 \_uiState.value = DashboardUiState.Error(exception.message)  
 }.collect { successState ->  
 \_uiState.value = successState  
 }  
 }  
 }  
}  
  
// Represents the different states of the Dashboard UI  
sealed interface DashboardUiState {  
 object Loading : DashboardUiState  
 data class Success(val user: User, val progress: LearningProgress) : DashboardUiState  
 data class Error(val message: String?) : DashboardUiState  
}

### 3. Component 2: The Backend Server

The backend is the brain of the application. It handles all business logic, manages the database, and acts as a secure gateway to the Gemini API.

* **Language:** Python
* **Framework:** FastAPI
* **Deployment:** Google Cloud Run (Serverless)

#### Key File Structure & Explanation:

/backend  
├── /app  
│ ├── /api  
│ │ ├── /v1  
│ │ │ ├── /endpoints // API route definitions (users, lessons, gemini)  
│ │ │ └── /schemas // Pydantic models for data validation  
│ ├── /core  
│ │ ├── config.py // Configuration settings  
│ │ └── security.py // Authentication and security logic  
│ ├── /services // Business logic (e.g., gemini\_service.py)  
│ └── main.py // FastAPI application entry point  
└── requirements.txt // Python dependencies

#### Source Code Snippet: AI Tutor API Endpoint (Simplified)

This snippet shows how the backend receives a user's message, crafts a prompt for the Gemini API, and returns the AI's response.

# /app/api/v1/endpoints/gemini.py  
  
from fastapi import APIRouter, Depends  
from app.core.security import get\_current\_user  
from app.services.gemini\_service import get\_conversational\_response  
from app.api.schemas import User, ChatMessage  
  
router = APIRouter()  
  
@router.post("/chat", response\_model=ChatMessage)  
async def chat\_with\_ai\_tutor(  
 message: ChatMessage,  
 current\_user: User = Depends(get\_current\_user)  
):  
 """  
 Endpoint for conversational practice with the AI Tutor.  
 """  
 # The gemini\_service module handles the complex logic of fetching  
 # chat history and crafting the perfect prompt for the Gemini API.  
 ai\_response\_text = await get\_conversational\_response(  
 user\_id=current\_user.id,  
 user\_message=message.text  
 )  
  
 return ChatMessage(text=ai\_response\_text, sender="ai")

### 4. Component 3: The Promotional Website

This is the public-facing showcase for the application. It's built as a single-page application (SPA) for maximum performance and a seamless user experience.

* **Languages:** HTML, CSS, JavaScript
* **Frameworks:** Tailwind CSS
* **Hosting:** Cloudflare Pages

#### Key File Structure & Explanation:

The entire website is contained within a single index.html file. The structure within that file is:

1. **HTML:** The semantic structure of the page.
2. **CSS (<style> block):** Custom styles that complement Tailwind CSS.
3. **JavaScript (<script> block):** All the logic for interactivity.

### 5. Manual Setup Guide (For Developers)

This section provides the step-by-step instructions for setting up the development environment for each component of the ThinkEdge application.

#### Prerequisites

* Git
* Python 3.9+ & Pip
* Android Studio (latest version)
* A Google Cloud account with Firestore and the Gemini API enabled.

#### A. Backend Setup

1. **Clone the Repository:**  
   git clone <repository\_url>  
   cd thinkedge/backend
2. **Create & Activate Virtual Environment:**
   * On macOS/Linux:  
     python3 -m venv venv  
     source venv/bin/activate
   * On Windows:  
     python -m venv venv  
     .\venv\Scripts\activate
3. **Install Dependencies:**  
   pip install -r requirements.txt
4. **Configure Environment Variables:**
   * Create a file named .env in the /backend directory.
   * Copy the contents of .env.example into it.
   * Fill in the required values, such as your GOOGLE\_API\_KEY and the path to your Firebase service account credentials JSON file.
5. **Run the Server:**  
   uvicorn app.main:app --reload  
     
   The backend server will now be running locally, typically at http://127.0.0.1:8000.

#### B. Android App Setup

1. **Open the Project:**
   * Launch Android Studio.
   * Select "Open" and navigate to the /app directory within the cloned repository.
2. **Configure Local Endpoint:**
   * In the project, find or create the local.properties file in the root of the Android project directory.
   * Add a line specifying the local backend URL:  
     BACKEND\_URL="http://10.0.2.2:8000/api/v1/"  
       
     *(Note: 10.0.2.2 is the special alias for the host machine's localhost when running in the Android emulator.)*
3. **Sync Gradle:**
   * Android Studio will likely prompt you to "Sync Now." Click it to download all the necessary dependencies for the project.
4. **Build and Run:**
   * Select an emulator or connect a physical device.
   * Click the "Run 'app'" button (green play icon) in the toolbar. Android Studio will build, install, and launch the application.

#### C. Promotional Website Setup

1. **Navigate to Directory:**  
   cd thinkedge/website
2. **View in Browser:**
   * Open the index.html file directly in your web browser.