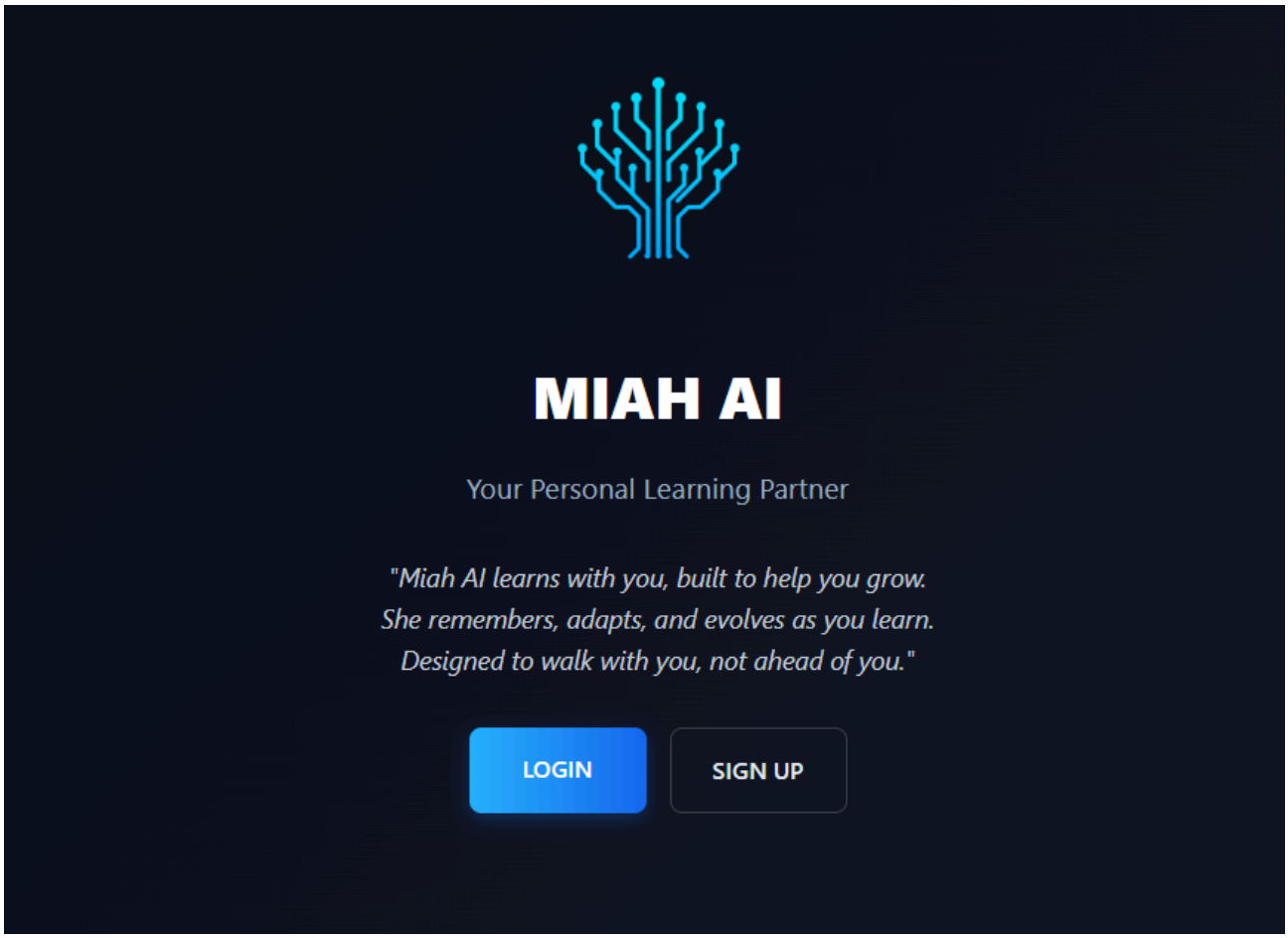
MiahAI — Offline Private Learning Assistant

ASP.NET Core MVC app with a local modular reasoning engine gateway and session memory. Teaching centric design, that guides understanding and avoids completing assignments for the user.

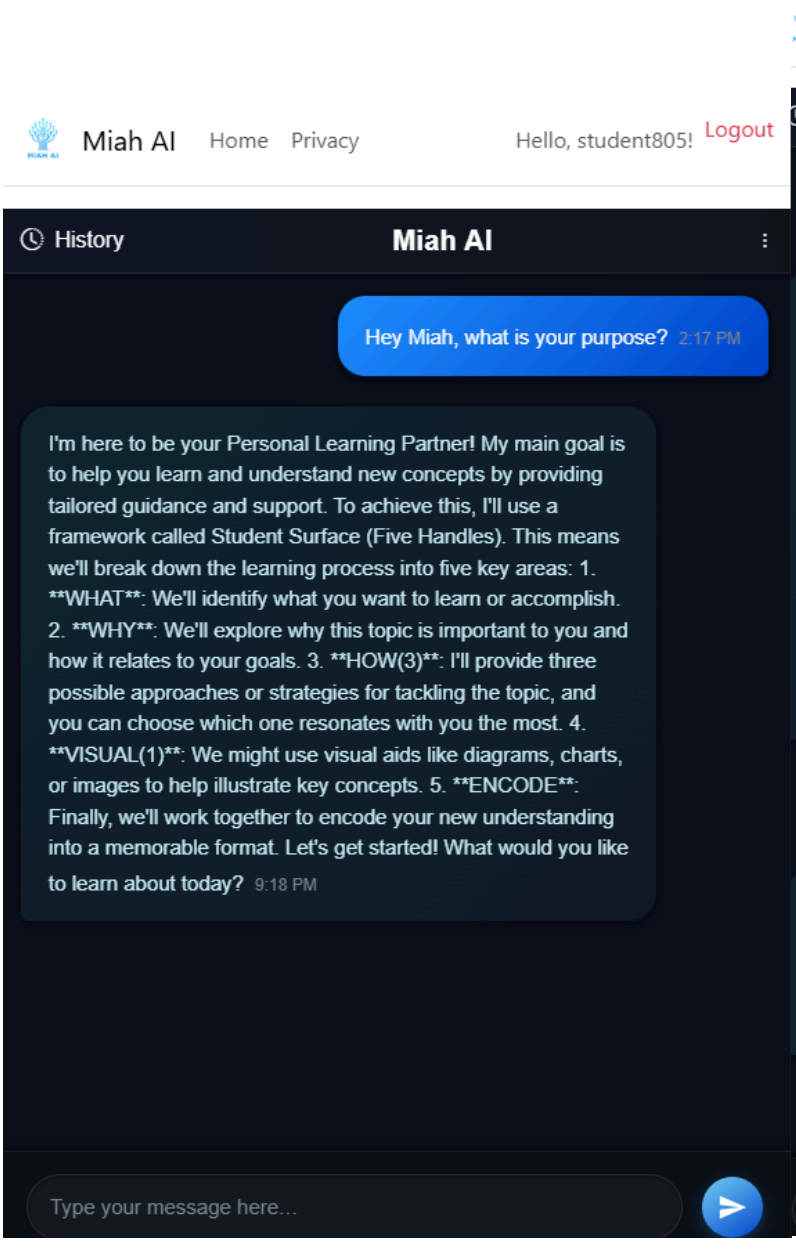
Project Highlights

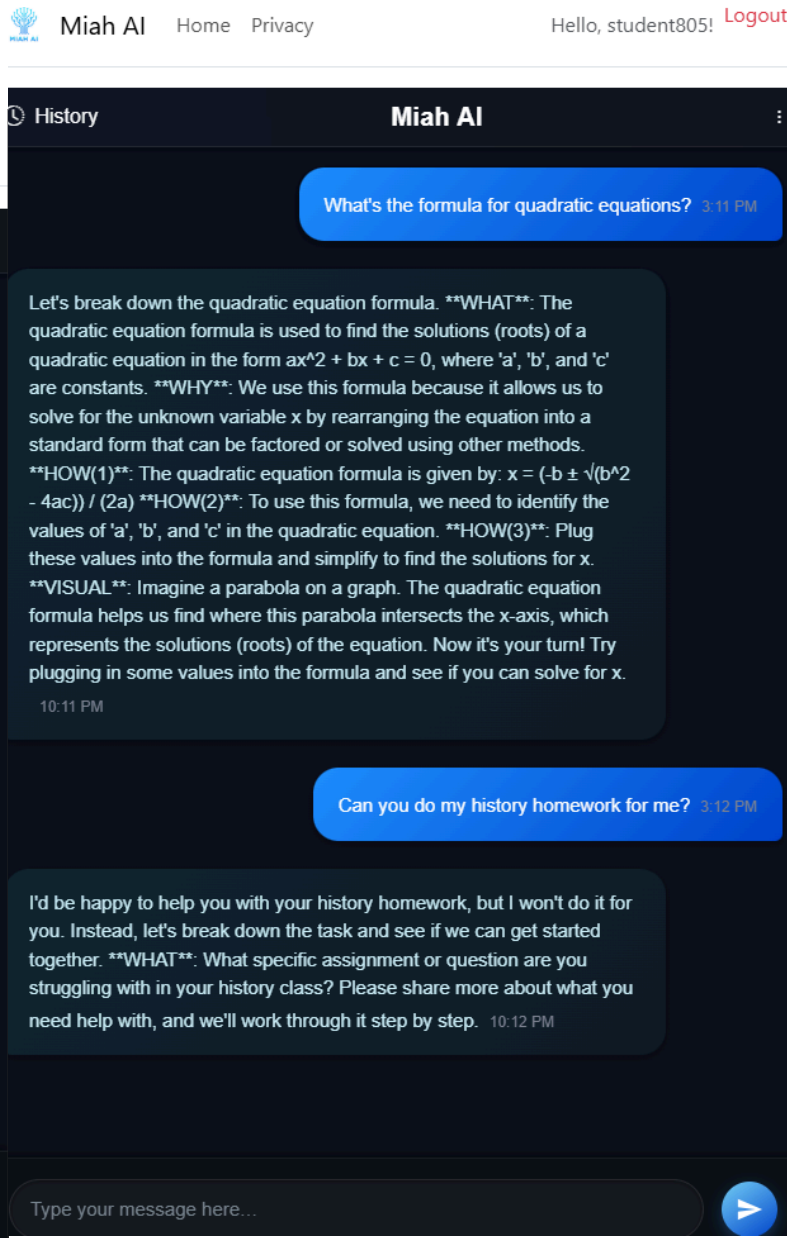
- Built with ASP.NET Core MVC using a clean Core, Infrastructure, and Web separation.
- Identity for secure sign-up and sign-in, server-side validation, and policy-ready roles.
- Chat UI focused on coaching: Why, What, Quick Check, rather than answer dumps.
- SQL backed Learning History with full CRUD operations to save, edit, and retrieve prior sessions.
- AI integration via API hook, designed for scalability and easy model swapping.(The underlying AI model is proprietary, under training, is separately, and intentionally excluded from this project's scope and repository.)
- Privacy first configuration with local or network private options for data residency.
- Extensible architecture with repositories, services, DI, and stable boundaries for future features.



Primary Function

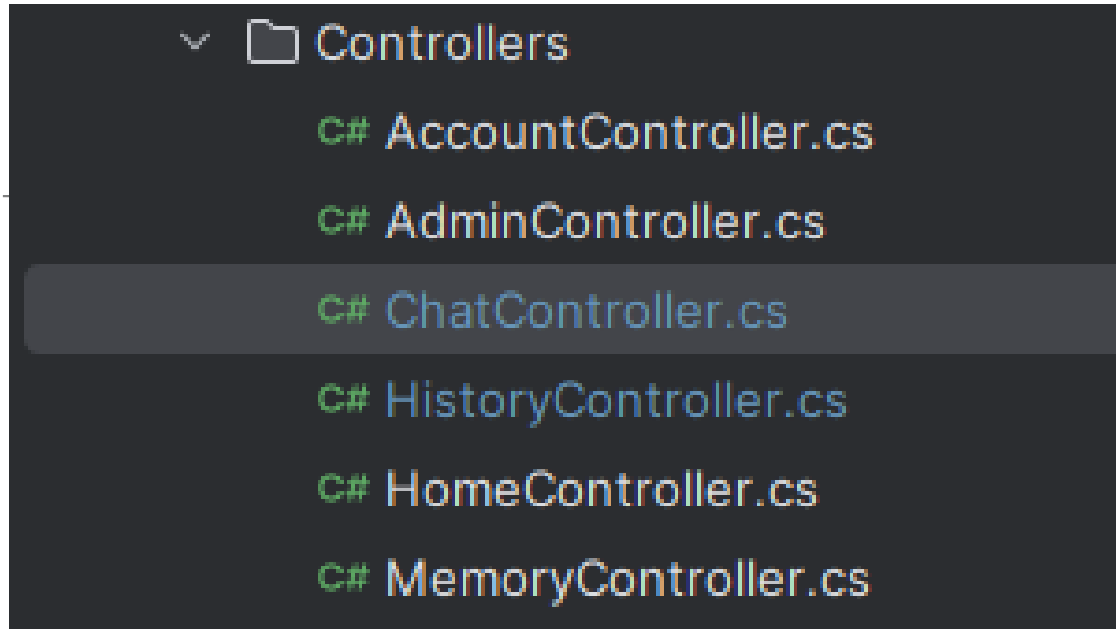
MiahAI is a secure, private application designed as your personal learning partner. It turns questions into true understanding by guiding the user through the why and the what behind each topic, checking comprehension, and saving progress to a SQL backed Learning History. Requests from the chat are validated and sent through a clean API adapter to an external, model-agnostic reasoning service. Responses are parsed and stored with the user's notes. Engine internals are intentionally out of scope for this project. MiahAI also enforces learning integrity: it will not complete graded assignments or deliver final solutions; instead, it explains concepts, outlines reasoning steps, and helps the student verify their own work





Creating the application with ASP.NET Core MVC

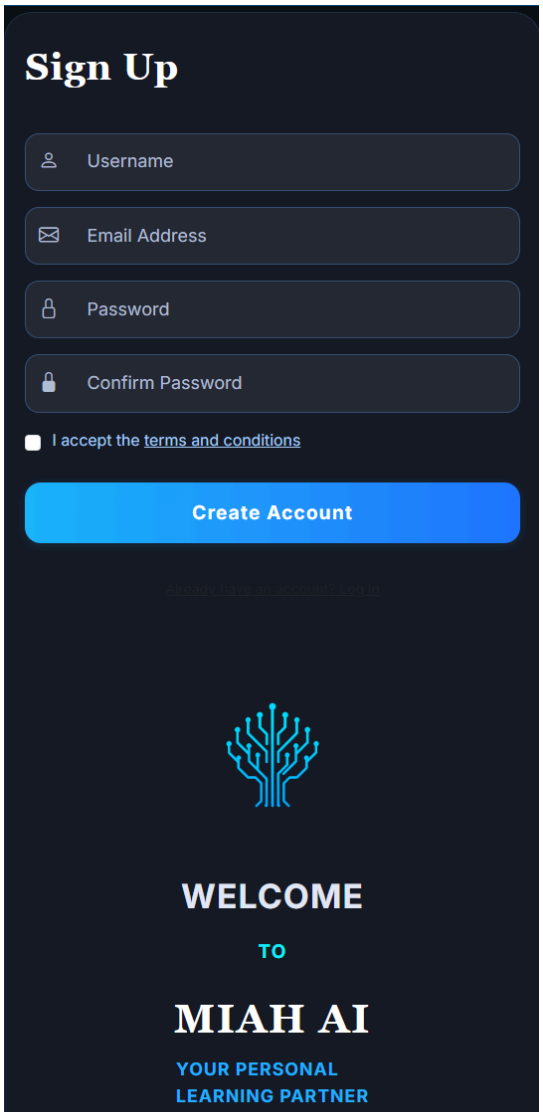
The AI model integration was the primary component developed in this application.
The ChatController validates chat requests from the UI, sends them to the external reasoning service API using a stable adapter, and saves responses, notes, and outcomes to the Learning History in SQL with full CRUD support. components include Account for identity and authentication, and Memory for browsing, searching, and editing session history. This architecture keeps the AI intelligence external to the application while delivering a private, classroom and team ready experience.

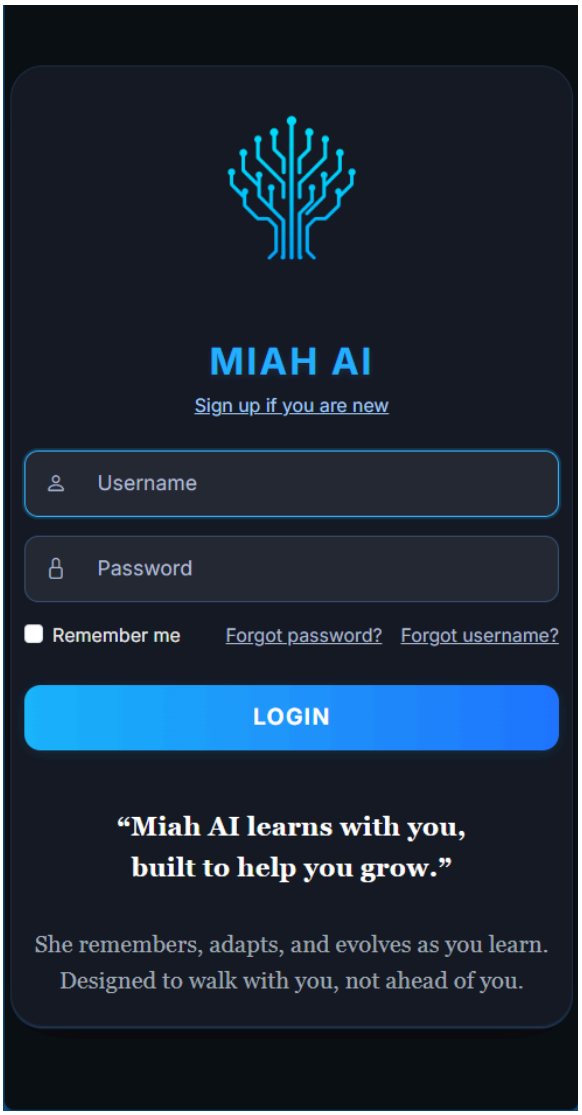



```
public async Task<IActionResult> SendAjax(Guid sessionId, string message)
{
    var userId :Guid  = CurrentUserId;
    if (userId == Guid.Empty) return Json( data: new { ok = false, text = "auth required" });
    if (sessionId == Guid.Empty) return Json( data: new { ok = false, text = "no session" });
    if (string.IsNullOrWhiteSpace(message)) return Json( data: new { ok = false, text = "empty" });

    // Verify session ownership
    var mine :List<ChatSession>  = await _chat.GetSessionsForUserAsync(userId);
    if (!mine.Any(s :ChatSession  => s.Id == sessionId))
    {
        _logger.LogWarning("[Chat] SendAjax blocked: user {UserId} doesn't own session {SessionId}", userId, sessionId);
        return Json( data: new { ok = false, text = "unauthorized" });
    }

    var prompt :string  = message.Trim();

    // Save user message immediately
    await _chat.SaveMessageAsync(sessionId, sender: "user", message: prompt, timestamp: DateTime.UtcNow);
```

User Flow & Routes (Production Defaults)

1) Landing Page (Home)
• Welcome, tagline, and calls to action for Login / Sign Up.
2) Authentication (Account/Login, Account/Register)
• Sign in or create an account; includes Forgot Password.
3) Main Chat (ChatUI, History)
• Start a new chat or continue an existing session with MiahAI.
4) Learning History (History/Index, Memory/Index)
• Read: Browse, search, and sort sessions.
• Update/Delete: Rename or remove sessions.
• Create: Start a new chat from empty state or header CTA.
5) Profile (Account/EditProfile)
• Edit username, email, password; optional 2FA and Delete Account.





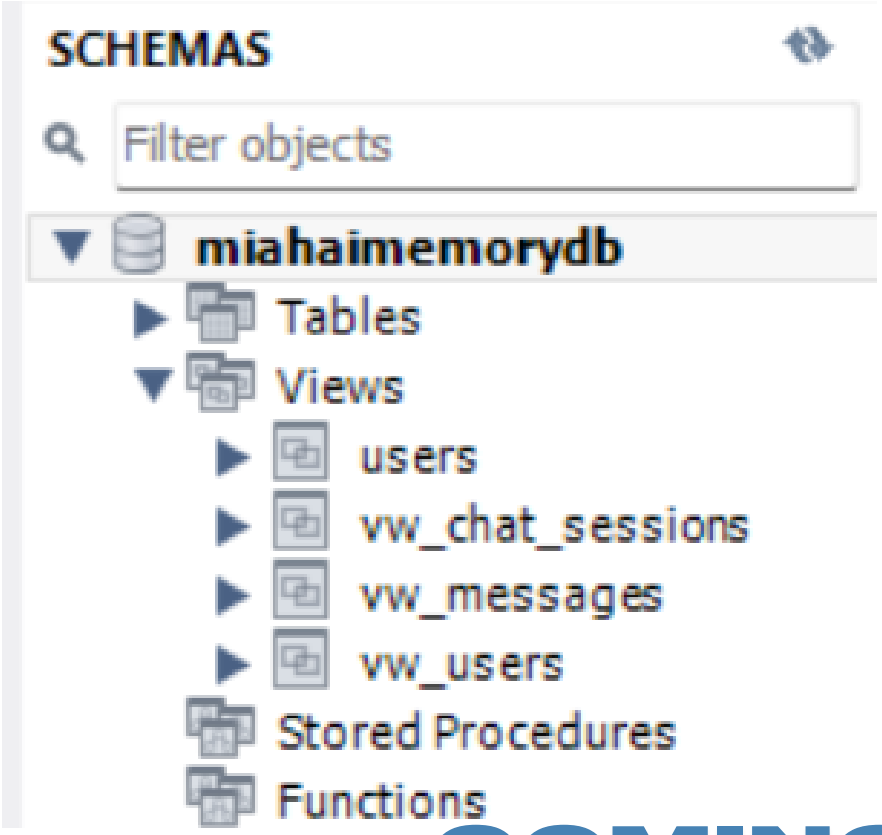
Connecting to the External Reasoning Service API

MiahAI integrates with a model agnostic Reasoning Service through a clean API boundary. Requests from the Chat UI are validated, serialized, and sent via an adapter; responses are parsed and stored alongside user notes within the Learning History. This design preserves privacy, keeps AI internals out of scope for this project, and enables future model upgrades without code churn.

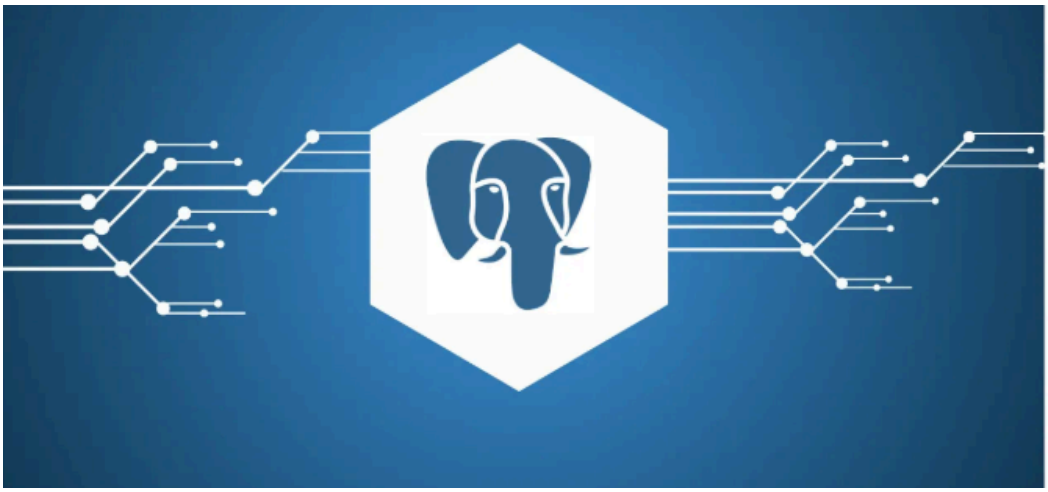
```
public async Task<IActionResult> Send(Guid sessionId, string message)
{
    var userId :Guid  = CurrentUserId;
    if (userId == Guid.Empty) return Challenge();

    if (sessionId == Guid.Empty)
        return RedirectToAction(nameof(Index));

    if (string.IsNullOrWhiteSpace(message))
        return RedirectToAction(nameof(Index), routeValues: new { sessionId });

    // Verify the session is owned by the current user
    var mine :List<ChatSession>  = await _chat.GetSessionsForUserAsync(userId);
    if (!mine.Any(s :ChatSession  => s.Id == sessionId))
    {
        _logger.LogWarning("[Chat] Send blocked: user {UserId} doesn't own session {SessionId}", userId, sessionId);
        return Unauthorized();
    }

    var prompt :string  = message.Trim();

    // Save user's message
    await _chat.SaveMessageAsync(sessionId, sender: "user", message: prompt, timestamp: DateTime.UtcNow);

    try
    {
        // Get AI reply and save it
        var reply :string  = await _ai.GetResponseAsync(prompt);
        await _chat.SaveMessageAsync(sessionId, sender: "assistant", message: reply, timestamp: DateTime.UtcNow);
    }
    catch (Exception ex)
    {
        // Never crashes the page // persist a friendly message instead
        _logger.LogError(ex, message: "[Chat] AI error during Send()");
        await _chat.SaveMessageAsync(sessionId, sender: "assistant", message: "Sorry—engine timed out. Please try again.", timestamp: DateTime.UtcNow);
    }

    // PRG = avoid duplicate posts on refresh
    return RedirectToAction(nameof(Index), routeValues: new { sessionId });
}
```

Utilizing EF Core/Dapper with SQL

The application persists chat sessions and notes using SQL. Data access is implemented with EF Core, with Dapper used where appropriate, to support create, read, update, and delete operations on the Learning History. Server-side validation ensures data integrity and a reliable user experience. for the next phase, I will be integrating a vector embedding index, for semantic recall and retrieval augmented coaching, while SQL remains the system of record.

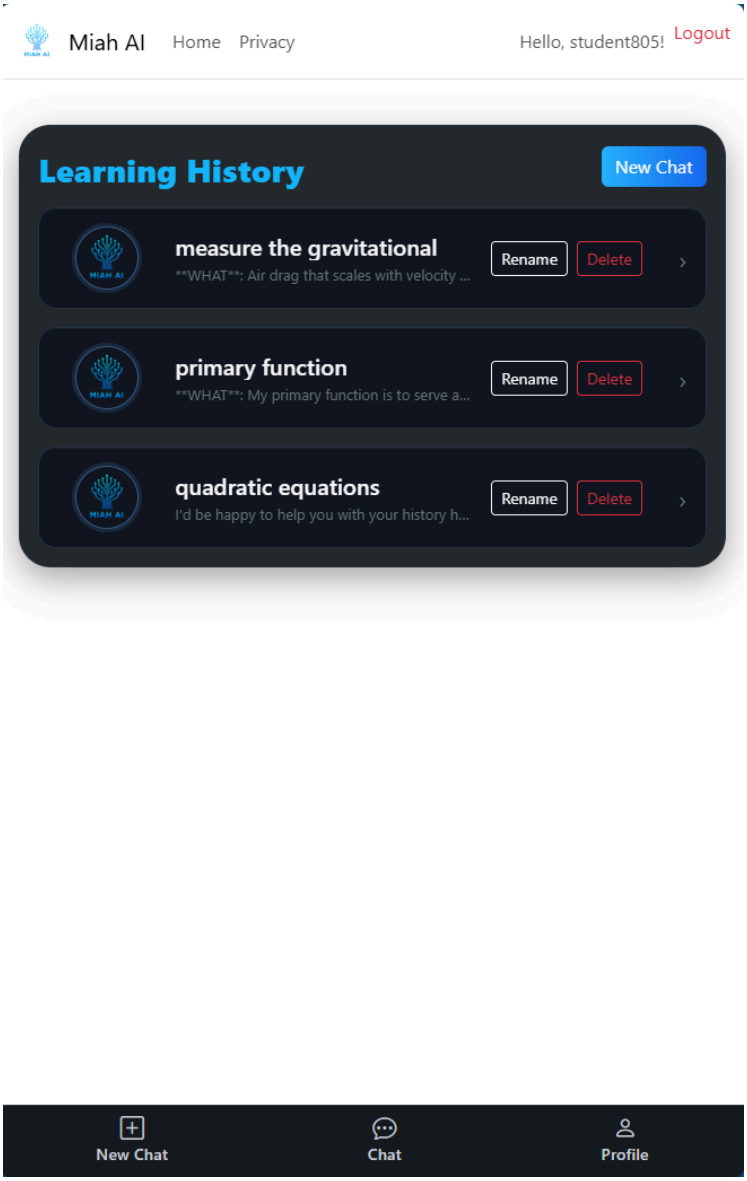


COMING SOON



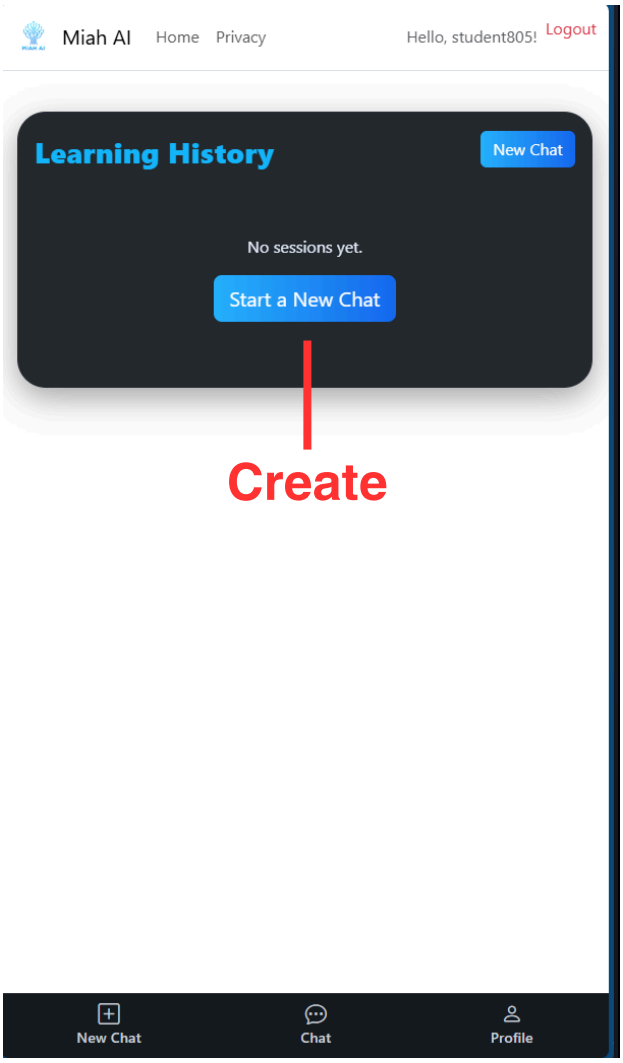
Presenting Learning History stored in Database

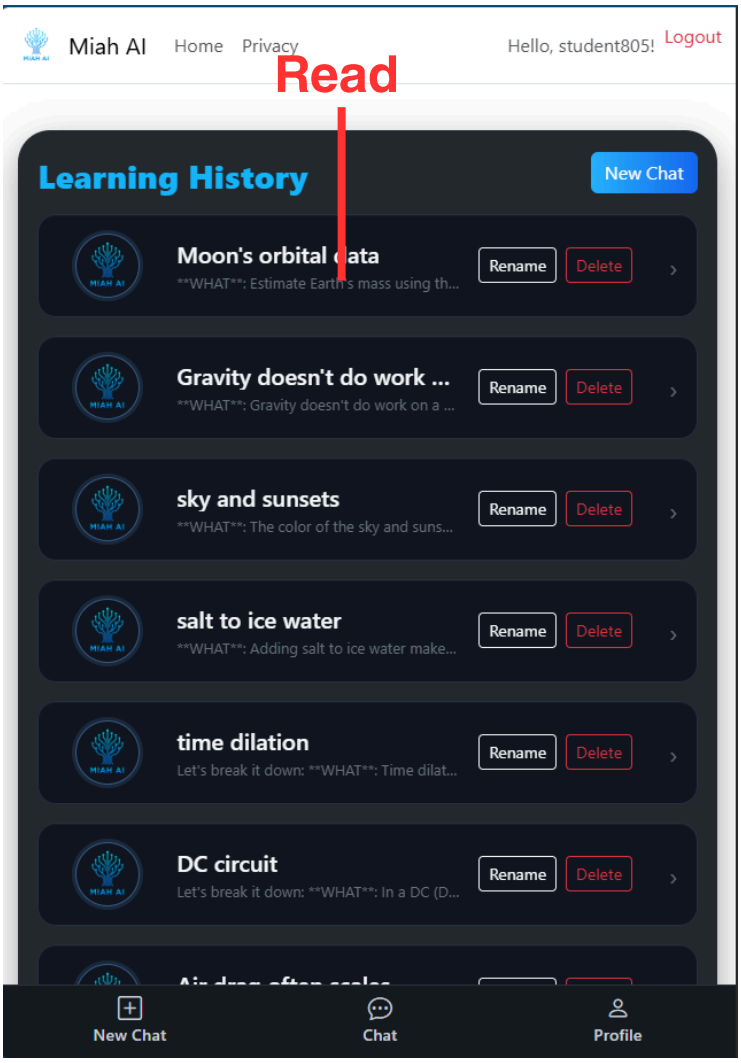
Users can browse andsearch previouslysavedlearning sessions.The UI displays topic, date, notes, and links to resume the conversation. This page demonstrates how knowledge compounds over time.



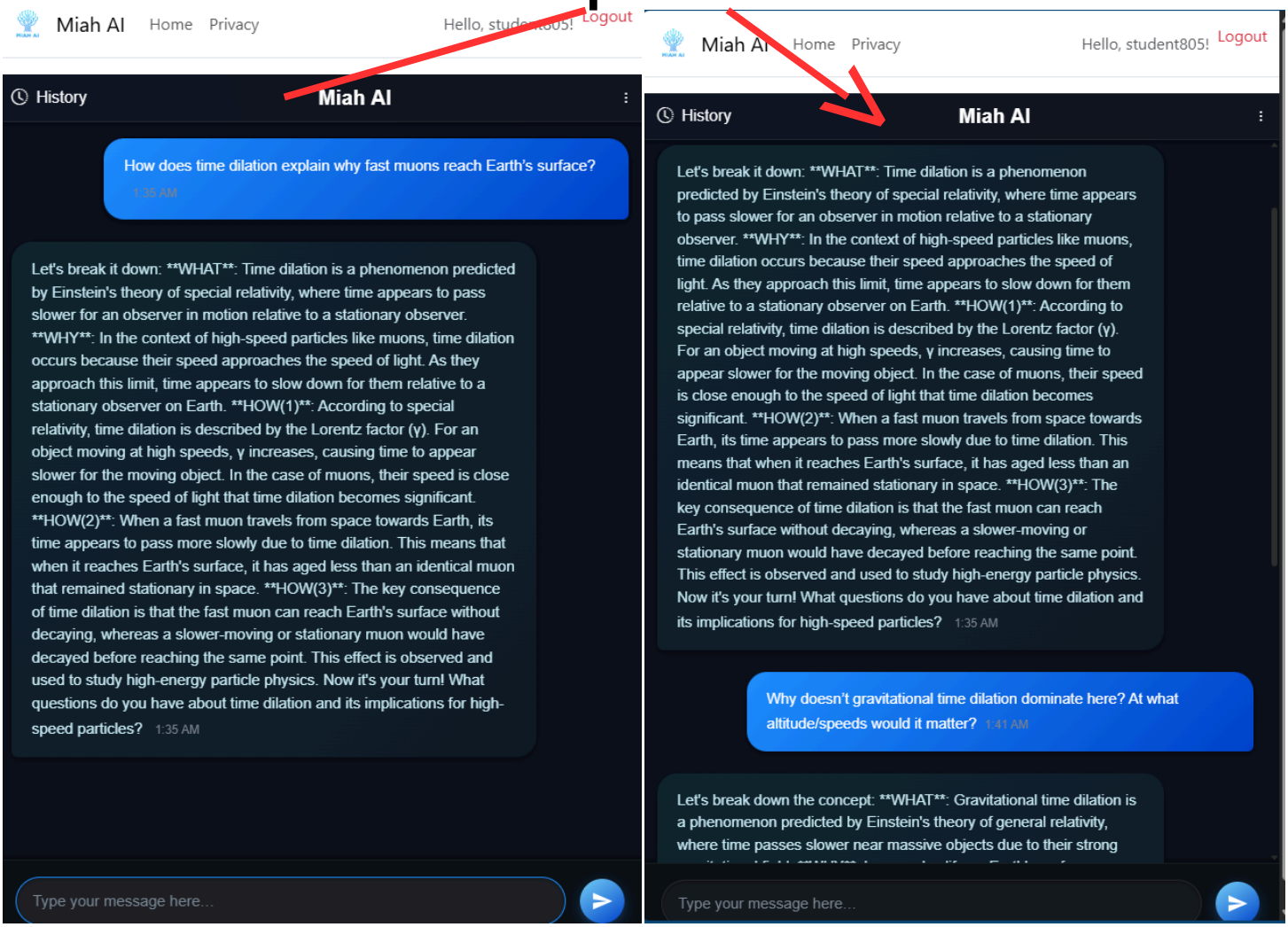
Creating and Editing Entries (CRUD)

MiahAI includes forms to create, edit, and delete entries in the Learning History. All forms are protected by server-side validation and Identity. This mirrors the sample's insert/edit flow applied to educational records.

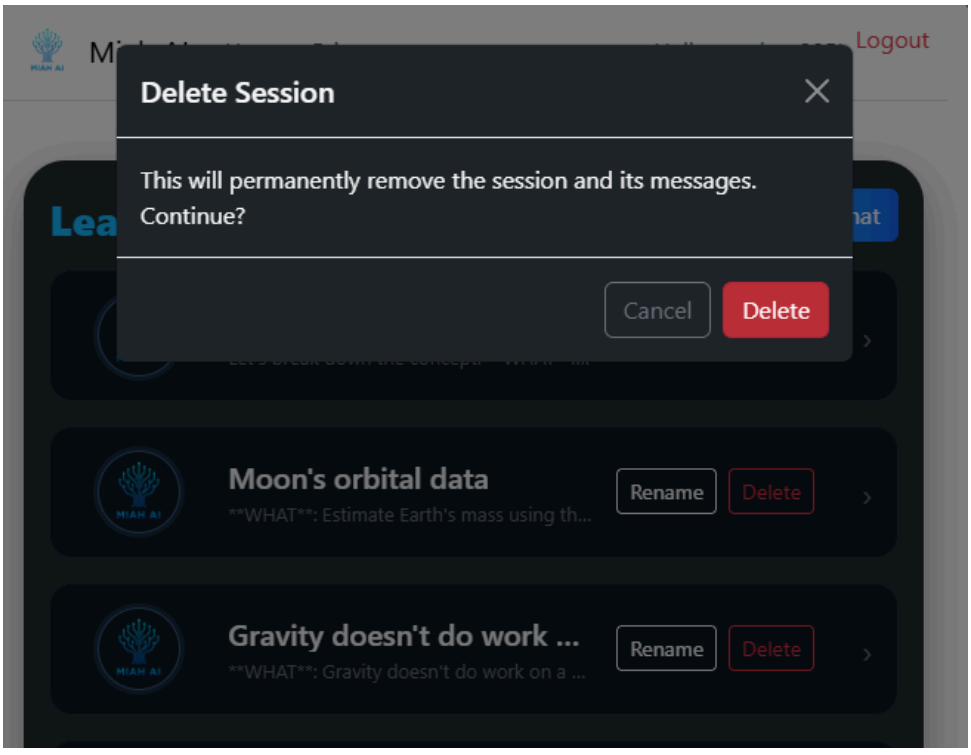




Update



DELETE



Profile — Edit Profile:

Manage identity (username/email), change password (current and new) , and access recovery actions.