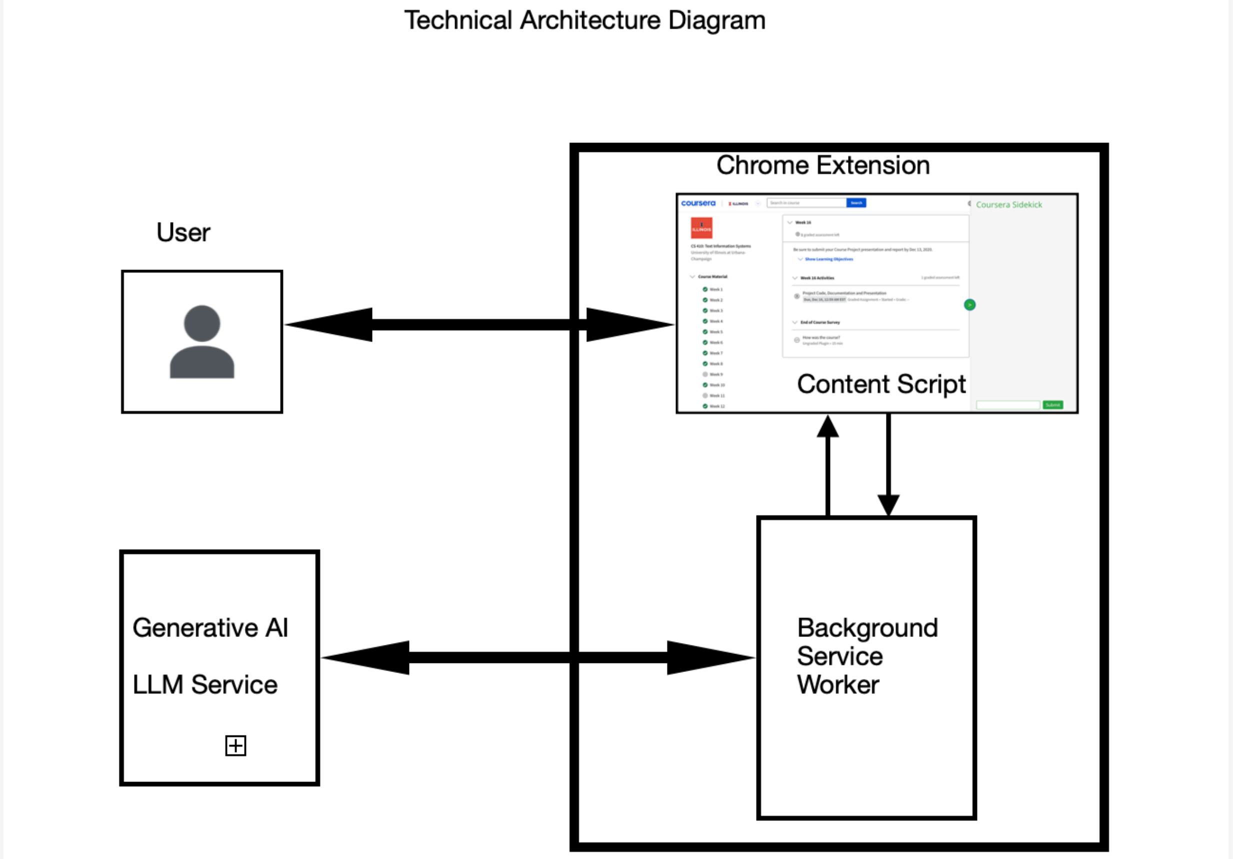
Project Name: Coursera Sidekick
Theme: Intelligent Browsing
Aim **Coursera Sidekick** is a Chrome extension aimed at enhancing the online learning experience on the Coursera platform. It leverages generative AI via the TOGETHER API, a managed service for large language models, to interpret and respond to user queries based on lecture content.
Project Purpose Picture this: you're swamped with lecture slides, articles, quizzes, and assignments. It's a race against time, and let's be real, it's tough to grasp everything, right? Now, imagine having a magic wand that transforms this chaos into clarity. Enter my Chrome Extension for Coursera i.e Coursera Sidekick. This little genius is your study buddy on steroids! Got a question? Just type it in. Confused by a lecture point? Just ask. Like a smart, helpful owl, it swoops in with concise, clear answers and key points. You'll be zipping through your studies, understanding more, and stressing less. This isn't just searching; it's intelligent browsing with a purpose. So, let's make studying not just quick, but also smart.
Coursera Sidekick Technical Architecture



Frontend:

Browser Extension UI (User Interface)

- `popup.html`: The HTML document providing the structure for the popup UI.
- `style.css`: The Cascading Style Sheets file defining the visual presentation of the popup UI.
- `popup.js`: The JavaScript file handling user interactions within the popup UI.

Content Script

Injected into the Coursera webpage, it interacts with the page content to extract information like lecture transcripts and user chat history.

Backend :

Extension Core

`background.js`: Manages background tasks, communicates with the content script, and sends requests to the server.

Server-Side Application

- **API Gateway**: Receives requests from `background.js` and routes them to the appropriate services.
- **Authentication Service**: Manages user authentication and permissions.
  - **Data Processing Service**:
  - **Text Parsing Module**: Extracts and preprocesses text data from transcripts and chat history.
  - **AI Query Processor (LAMM 7B)**: Processes user queries using the LAMM 7B model to generate contextually relevant answers.
- **Summary and Topic Extraction Service**: Utilizes NLP (Natural Language Processing) techniques to summarize content and extract main topics.
- **MathJax and Markdown Rendering Service**: Converts LaTeX and markdown content into readable and presentable formats.

Data Storage

- **User Data Store**: To save API service key
- **Cache**: Temporarily stores frequently accessed data like chat history and lecture content for quick retrieval.

Integration

- **TOGETHER API**: An external API providing the generative AI LLM capabilities.
- **MathJax.js**: A library integrated into the frontend for rendering mathematical equations.
- **Marked.min.js**: A library used in the frontend for parsing and displaying markdown text.

Deployment

- **Extension Hosting**: The Chrome Web Store hosts the extension package consisting of `manifest.json`, HTML, CSS, and JS files.
- **Server Hosting**: Used managed LLM Service.

This architecture ensures a seamless and responsive user experience, allowing the Coursera Sidekick to provide intelligent and quick assistance to learners navigating through course materials on Coursera.

Flow of Operations

- The user interacts with the Coursera webpage and has questions or needs a summary of the content.
- The user clicks the extension icon, which opens `popup.html`.
- User inputs are captured by `popup.js` and sent to `background.js`.
- `background.js` sends the user query and contextual data to the server-side application via the API Gateway.
- The user is authenticated, and their query is processed by the AI Query Processor using the LAMM 7B model.
- Simultaneously, the Summary and Topic Extraction Service processes the lecture content.
- The MathJax and Markdown Rendering Service formats any mathematical or markdown content included in the response.
- The server responds with the requested information, which is then displayed to the user through the popup UI.

Programing Language

During project proposal I planned to use Python as programming language however, As a developer, there are several pragmatic reasons for choosing JavaScript (JS) and JSON for the Coursera Sidekick Chrome extension over Python, especially considering the nature of browser extensions and the technical requirements of the project

- **Browser Compatibility**: JavaScript is the native scripting language for browsers. Browser extensions are typically written in HTML, CSS, and JavaScript, which are the core technologies of the web. This makes JavaScript an immediate choice for any browser extension, including the Coursera Sidekick.
- **Real-time Interaction**: JavaScript excels in real-time content manipulation on web pages, which is essential for a Chrome extension like Coursera Sidekick that interacts directly with web content, such as modifying the DOM to display summaries or interact with the Coursera platform's interface.
- **JSON Integration**: JSON (JavaScript Object Notation) is a lightweight data-interchange format that is easy for humans to read and write and easy for machines to parse and generate. Given that JSON is a subset of JavaScript, it integrates seamlessly with JS, allowing for efficient data exchange between the extension and the server-side API.
- **Event-Driven Architecture**: Chrome extensions rely heavily on an event-driven architecture, which JavaScript naturally supports. This is important for responding to user actions, such as clicks or keyboard inputs, in a responsive and non-blocking manner.
- **Portability and Hosting**: JavaScript, along with HTML and CSS, can be packaged into a single Chrome extension file and uploaded to the Chrome Web Store. This is not as straightforward with Python, as it would require setting up a server to run the Python code, which adds complexity and potential latency.
- **Performance**: JavaScript's performance in the context of a browser extension is generally superior to Python for client-side execution. Since extensions often require immediate response and low latency, JavaScript's speed in the web environment is a significant advantage.
- **Large Language Model APIs**: The TOGETHER API and other AI services can be called using JavaScript via asynchronous HTTP requests (AJAX), which is a standard practice in web development. This allows the extension to utilize AI capabilities without the need to implement and maintain the AI logic within the extension itself.

While Python is a powerful language with its own strengths, for the client-side execution required by a browser extension, JavaScript is the more suitable choice. The decision to use JavaScript and JSON aligns with the industry standards for web-based applications and Chrome extension development, ensuring the Coursera Sidekick is efficient, secure, and user-friendly.

Techniques Used

In order to achieve aim of the project inline with intelligent browsing theme I choose to use a generative AI with a Large Language Model (LLM) like LAMM 7B for the Coursera Sidekick over the approach of semantic search using text embeddings like BERT or Sentence-BERT coupled with similarity measures such as cosine similarity.

- **Contextual Understanding**: LLMs have been trained on a diverse range of internet text. This enables them to understand and generate responses that are contextually rich and nuanced. While BERT and Sentence-BERT can understand sentence structure and context to a degree, LLMs can often generate more coherent and contextually relevant answers.
- **Generative Capabilities**: LLMs can generate new text that doesn't exist in the source materials. This is particularly useful for summarizing content and providing explanations that aren't directly found in the lecture transcripts or materials. Text embeddings and cosine similarity are excellent for finding existing similar content but do not generate new text.
- **User Interaction**: The generative aspect of LLMs allows for a more conversational and interactive experience. Users can ask follow-up questions, and the LLM can generate answers that build on previous interactions. This creates a more dynamic and engaging user experience compared to the static nature of similarity searches.
- **Ease of Integration**: The TOGETHER API and other LLM-based services provide easy-to-integrate endpoints that handle all the complexity of the AI model. This means you can leverage the power of LLMs without the overhead of developing, training, or maintaining complex models like BERT.
- **Real-Time Performance**: While both LLMs and semantic search techniques require processing time, generative AI models provided as managed services are optimized for speed and scale, ensuring real-time performance that is crucial for the user experience in an interactive tool like Coursera Sidekick.
- **Broader Understanding**: LLMs are typically trained on a broader corpus of data than BERT models, which often specialize in specific domains. This broad training allows LLMs to have a wide-ranging understanding, which is beneficial for a platform like Coursera that covers a vast array of subjects.

While text embeddings and similarity measures provide precise results for finding content that matches the search query closely, the generative AI using LLM offers a comprehensive solution that not only finds relevant information but also explains, summarizes, and interacts with the user in a more human-like manner. This aligns with the goal of creating an intelligent assistant that enhances the learning experience on Coursera by providing immediate, relevant, and accessible assistance.

Code Files Details

- content.js
- background.js
- manifest.json
- marked.min.js
- MathJax.js
- popup.html
- popoup.js
- styles.css
- transformers.js
- Images