Lecture 08: Working with ChatClient

Why ChatClient?

- Earlier, we worked with **ChatModel**, which lets us interact with LLMs but has limited features.
- **ChatClient** is a higher-level, fluent API built on top of ChatModel.
- Provides more flexibility:
    - Send prompts and receive responses easily.
    - Responses can be **single messages** or **streamed**.
    - Access to more metadata (not just text).

How to Implement:

- **ChatClient is built on ChatModel**
    - You still need a ChatModel instance to create it.
    - Example: ChatClient.create(chatModel).
- **Creating ChatClient**

```
private ChatClient chatClient;

public OpenAIController(OpenAiChatModel chatModel) {
    this.chatClient = ChatClient.create(chatModel);
}
```

- **Using Prompts**
    - **Unlike chatModel.call(), here we use:**
        - **chatClient.prompt(message) → prepares a request.**
        - **.call() → sends request to LLM.**
        - **.content() → extracts the plain text response.**

Advantages:

- Cleaner and more readable API.
- Can fetch not only content but also **chat responses with metadata**.
- Supports structured outputs in the future.

Overall Code:

```java
@RestController
public class OpenAIController {
  private ChatClient chatClient;

  public OpenAIController(OpenAiChatModel chatModel) {
      this.chatClient = ChatClient.create(chatModel);
  }

  @GetMapping("/api/{message}")
  public ResponseEntity<String> getAnswer(@PathVariable String message)
{    String response = chatClient.prompt(message)
              .call()
              .content();
      return ResponseEntity.ok(response);
  }
}
```

Key Points:

- **ChatModel vs ChatClient**
  - ChatModel.call("msg") → quick, direct call.
  - ChatClient.prompt("msg").call().content() → more flexible and extendable.
- **ResponseEntity is** used for better practice in Spring and allows status codes along with the response body.
- With ChatClient, you now have more power and flexibility compared to plain ChatModel.