**Project Book: Website AI Assistant (WAS) & Agentic Chatbot**

**1. Introduction**

In the last few years, the digital landscape has undergone a seismic shift, driven by advancements in artificial intelligence. Users now expect more than just static information from websites; they demand instant, accurate, and interactive engagement. The traditional model of navigating complex menus or waiting for human support is quickly becoming obsolete. This new era calls for intelligent solutions that can understand user intent and provide immediate, context-aware assistance, transforming passive browsing into a dynamic conversation.

This necessity to innovate is the driving force behind our work. We recognized a critical gap between the vast amount of information stored on websites and the ability of users to access it efficiently. To bridge this gap, we have developed two distinct but complementary projects, each exploring a different facet of conversational AI’s potential.

Our primary focus has been the development of the **Website AI Assistant (WAS)**, a sophisticated platform built on the principle of Retrieval-Augmented Generation (RAG). The core mission of WAS is to empower any organization to create and deploy a chatbot that is a true expert on its own domain. By ingesting and understanding the content of a given website—from product descriptions and support articles to policy documents—the WAS platform generates a chatbot capable of answering user queries with high fidelity and accuracy. It effectively turns a website’s content into a structured, queryable knowledge base. This allows businesses to offer 24/7 customer support, guide users to relevant information, and significantly enhance user engagement without the need for extensive manual programming or rule-setting. The platform provides a comprehensive solution, including a user-friendly dashboard for chatbot creation and customization, a robust backend for data processing, and a lightweight, embeddable widget that integrates seamlessly into any existing website.

Complementing the informational power of WAS, we also explored the frontier of task-oriented AI with a minor, yet significant, project: an **Agentic Chatbot**. This project serves as a proof-of-concept for a more advanced, action-driven assistant. Unlike the RAG chatbot, which primarily retrieves and synthesizes information, the agentic chatbot is designed to perform tasks. Built using an Agent Development Kit, our prototype is a customer service agent for a retail environment. It can interact with mocked backend systems to manage a user’s shopping cart, provide personalized product recommendations, and even schedule appointments. This project demonstrates the potential for AI to move beyond answering questions and become an active participant in a user’s journey, capable of executing complex, multi-step operations.

Together, these two projects represent a comprehensive exploration of modern conversational AI. The Website AI Assistant provides a scalable, practical solution for automated information delivery, while the Agentic Chatbot showcases the future of interactive, task-oriented AI. This document will provide a detailed account of the theoretical foundations, design, architecture, and implementation of both systems, offering a complete project book for these innovative endeavors.

**2. Theoretical Background**

The development of the Website AI Assistant (WAS) and the complementary Agentic Chatbot is grounded in a modern, robust client-server architecture and leverages several cutting-edge artificial intelligence paradigms. This section details the foundational theories upon which these systems are built.

**2.1 Architectural Foundation**

Both projects are built upon a client-server model, which separates the user-facing interface (client) from the data processing and business logic (server). This separation is critical for creating a scalable, maintainable, and secure application.

The **Website AI Assistant (WAS)** employs a sophisticated three-tier architecture, a proven software design pattern that separates the application into logical and physical computing tiers. This separation enhances scalability, maintainability, and flexibility by decoupling the user interface, business logic, and data storage layers.

1. **Presentation Tier (Client):** This tier consists of two distinct clients—a web-based administrative dashboard and a lightweight, embeddable widget for end-users. It is responsible for rendering the user interface and capturing user input.
2. **Logic Tier (Server/Backend):** This is the authoritative core of the system. It executes the business logic, handles data processing (such as website ingestion), manages asynchronous tasks, and orchestrates all interactions with AI models and other services.
3. **Data Tier:** This tier includes both a traditional database for storing application state (like user and chatbot configurations) and a specialized vector database for storing the indexed knowledge required for the RAG process.

The **Agentic Chatbot** also follows a client-server model, but its architecture is conceptually defined by the principles of agent-based systems. The server-side logic encapsulates the agent’s core reasoning loop, state management, and its ability to interact with a set of defined tools, representing a distinct architectural pattern focused on autonomous task execution.

**2.2 Core AI Paradigm: Retrieval-Augmented Generation (RAG)**

The central theory behind the WAS platform is **Retrieval-Augmented Generation (RAG)**. Traditional Large Language Models (LLMs) generate responses based solely on the vast, but static, data they were trained on. This can lead to outdated or generic answers. RAG enhances LLMs by connecting them to external, up-to-date knowledge sources.

The process works as follows:

1. **Ingestion & Indexing:** Website content (HTML, PDFs, etc.) is collected, parsed, and broken down into smaller, manageable chunks of text. Each chunk is then converted into a numerical representation (a vector embedding) and stored in a specialized vector database. This index is optimized for fast and efficient similarity searches.
2. **Retrieval:** When a user asks a question, the system first converts the query into a vector embedding. It then searches the vector database to find the text chunks that are most semantically similar to the user’s query.
3. **Augmentation & Generation:** The retrieved text chunks (the “context”) are combined with the original user query into a detailed prompt. This augmented prompt is then sent to an LLM (like Google’s Gemini). The model uses the provided context to generate a relevant, accurate, and factually grounded response, rather than relying only on its internal knowledge.

This RAG pipeline is the key to making the WAS chatbot a true expert on a specific website’s content.

**2.3 Core AI Paradigm: Agentic Models & Tool Use**

The agentic chatbot project is built upon the theory of **tool-using agents**. This paradigm extends the capabilities of LLMs beyond text generation, turning them into reasoning engines that can interact with external systems to accomplish tasks.

The core concept involves providing the LLM with a set of “tools” (i.e., functions or APIs) and their descriptions. When a user makes a request, the LLM’s reasoning capabilities are used to:

1. **Deconstruct the Request:** Understand the user’s intent and the steps required to fulfill it.
2. **Select the Appropriate Tool:** Based on the descriptions, choose the correct tool or sequence of tools to execute.
3. **Extract Parameters:** Identify and extract the necessary arguments for the chosen tool from the user’s query.
4. **Execute and Observe:** Call the tool and receive its output.
5. **Synthesize a Response:** Formulate a human-readable response based on the outcome of the tool execution.

This approach allows the agent to perform actions in the real world, such as accessing a database, calling an external API, or modifying a user’s state in a system, making it a powerful model for creating interactive, task-oriented assistants.

**5. Scenarios**

This section outlines typical user stories and system processes, illustrating how users interact with the platform and how the system responds to those interactions.

**5.1 Website AI Assistant (WAS) Scenarios**

**5.1.1 User Story: Administrator Creates and Deploys a New Chatbot**

**Actor:** A website administrator.

**Goal:** To create a new chatbot, train it on the company’s public FAQ page, customize its appearance, and deploy it to the website.

**Process:**

1. **Login:** The administrator navigates to the WAS platform’s web dashboard and logs in with their credentials.
2. **Create Chatbot:** From the main dashboard, the administrator clicks “Create New Chatbot.” They give the chatbot a name, “Company FAQ Bot,” and save it.
3. **Add Data Source:** The administrator navigates to the “Data Sources” tab for the new chatbot. They select the “URL” option and paste the web address of their company’s FAQ page (e.g., https://mycompany.com/faq).
4. **System Ingestion Process:**
   * The backend receives the request and queues an asynchronous ingestion task using Celery.
   * A web crawling worker fetches the HTML content from the provided URL.
   * The system parses the HTML, extracts the main textual content, and cleans it.
   * The cleaned text is segmented into smaller, semantically meaningful chunks.
   * Each chunk is passed to a text embedding model, which converts it into a vector.
   * These vectors are stored and indexed in a specialized vector database, associated with the “Company FAQ Bot.”
   * The dashboard updates to show the FAQ page as a successfully ingested data source.
5. **Customize Appearance:** The administrator goes to the “Customization” tab. They change the widget’s primary color to match their company’s branding and update the “Launcher Text” to say “Ask us anything!”
6. **Test the Chatbot:** The administrator uses the built-in chat interface in the dashboard to test the chatbot. They ask, “What are your business hours?” The system retrieves the relevant context from the ingested FAQ page and generates an accurate answer.
7. **Deploy to Website:** Satisfied with the result, the administrator copies the provided JavaScript snippet and embeds it into their website’s HTML. The “Company FAQ Bot” launcher now appears on their live site, ready to assist visitors.

**5.1.2 User Story: End-User Asks a Question via Voice**

**Actor:** A visitor on the company’s website.

**Goal:** To find out the company’s return policy without having to search the website manually.

**Process:**

1. **Initiate Interaction:** The visitor sees the “Ask us anything!” launcher on the website and clicks it, opening the chat widget.
2. **Start Voice Input:** The visitor clicks the microphone icon in the chat widget. The browser prompts for microphone access, which the user grants.
3. **Ask Question:** The visitor speaks their question: “What is your return policy?”
4. **System Voice-to-Text Process:**
   * The widget’s JavaScript captures the audio stream.
   * The audio data is sent to the WAS backend’s speech-to-text endpoint.
   * The backend uses a cloud-based STT service (e.g., Google Cloud Speech-to-Text) to transcribe the audio into the text “What is your return policy?”.
   * The transcribed text is returned to the widget and displayed in the chat window.
5. **System RAG Process:**
   * The backend receives the transcribed query.
   * The RAG service converts the query into a vector and searches the vector database for the most relevant text chunks from the ingested company documents.
   * The top-matching chunks, which contain details about the return policy, are retrieved.
   * These chunks are combined with the user’s query into a prompt for the LLM.
   * The LLM generates a concise answer summarizing the return policy based on the provided context.
6. **Receive Answer:** The generated text answer is displayed in the chat widget. Because the user initiated the query with voice, the system also sends the text to a TTS service to generate an audio version of the response.
7. **Audio Playback:** The audio response is streamed back to the widget, which automatically plays it for the user. The user sees the text answer and hears the spoken response simultaneously.

**5.2 Agentic Chatbot Scenario**

**5.2.1 User Story: Customer Modifies an Order and Schedules a Service**

**Actor:** A returning customer of a home and garden store.

**Goal:** To get advice on a recent purchase, modify their current shopping cart, and schedule a planting service.

**Process:**

1. **Initiate Conversation:** The customer starts a conversation with the agent. The system recognizes the customer as “Alex” based on their (mocked) logged-in state.
2. **Agent Greeting & Context:** The agent greets the customer by name and acknowledges items already in their shopping cart, demonstrating awareness of the user’s state.
3. **Customer Inquiry:** The customer explains they are unsure if they have the right soil for a plant they recently purchased.
4. **Agent Tool Use (Video Identification):**
   * The agent recognizes the ambiguity and determines it needs more information.
   * **Tool Selected:** send\_call\_companion\_link
   * The agent calls the tool to send a secure video link to the customer’s phone.
   * The customer uses the link to show the plant to the agent. The agent (simulating multimodal input) identifies the plant.
5. **Agent Tool Use (Recommendation & Cart Modification):**
   * Based on the plant identification, the agent determines a better soil is available.
   * **Tool Selected:** get\_product\_recommendations
   * The agent recommends the appropriate soil and a suitable fertilizer.
   * The customer agrees to the change.
   * **Tool Selected:** modify\_cart
   * The agent calls the tool to remove the incorrect soil from the cart and add the recommended soil and fertilizer.
6. **Agent Tool Use (Upselling & Scheduling):**
   * The agent suggests a professional planting service for the new items.
   * The customer expresses interest and asks for available times.
   * **Tool Selected:** get\_available\_planting\_times
   * The agent retrieves and presents a list of open appointment slots.
   * The customer chooses a time.
   * **Tool Selected:** schedule\_planting\_service
   * The agent books the appointment and confirms the details with the customer