LAB 1

Analysis and Identification of the Suitable Process Models – SDLC

Name: Yash Aneja

Reg No: 23BCE1918

Project Nature:

This project is a NoCode chatbot builder platform where users can create and embed custom chatbots into their websites without writing any code.

Users will define a chatbot's role, upload their website's documentation, and the system will automatically generate vector embeddings from the documents, store them securely, and use a Gemini LLM to answer user queries.

A unique JavaScript code snippet will be generated for each chatbot, allowing seamless embedding into any website.

It includes:

- Role-based chatbot creation with custom instructions.
- Document parsing & embedding generation for knowledge-based responses.
- Multi-tenant storage to handle multiple users securely via Supabase.
- Customizable chatbot widget with drag-and-drop positioning & live preview.
- Modern animated UI with 3D elements for professional branding.
- Real-time chat interface powered by Gemini for accurate, context-aware responses.

Key Points:

- Multi-module system: User authentication, chatbot role setup, document upload, embeddings generation, chatbot preview customization, JavaScript snippet generation, and real-time chat responses.

- Requirements partially fixed (core role definition, document upload) but will evolve (advanced UI customization, analytics, multi-language support).

- Continuous improvement possible by collecting chatbot usage analytics and feedback.

- Integration of multiple technologies — Vite + Tailwind + Framer Motion for frontend, Supabase for backend & database, Gemini API for LLM responses, and vector embeddings for RAG.

Suitable SDLC Process Model:

Incremental Model

Reason for Selection:

- Project modules (auth, chatbot role definition, document embedding, chatbot preview, snippet generation, chat API) can be developed in increments.

- Allows early functional release (basic chatbot without customization) while advanced features (drag-and-drop preview, analytics, 3D animations) are added later.

- Supports continuous feedback from early adopters to improve UI/UX and chatbot performance.

- Fits well for projects where core requirements are known but customization and extra features evolve with user needs.

How the Incremental Model Fits This Project:

First Increment:

- Implement authentication (Supabase) and dashboard.

- Create chatbot role definition form.

- Generate a basic chatbot using Gemini API (no document upload).

Second Increment:

- Add document upload & parsing (PDF, DOCX, TXT).

- Generate embeddings & store in Supabase vector store.

- Integrate chatbot with document-based answers (RAG).

Third Increment:

- Add drag-and-drop chatbot preview builder.
- Implement customization options (avatar, colors, fonts).

Fourth Increment:

- Generate unique JavaScript snippet for embedding chatbot in any website.
- Add live website preview mode.

Final Increment:

- UI/UX enhancements with 3D elements & animations.
- Add analytics & chatbot performance tracking.
- Optimize security & scalability.

Advantages for This Project:

- Early deployment of a working chatbot platform.
- Feedback-driven development for UI and features.
- Parallel development possible (frontend, backend, AI).
- Lower risk as each module is tested in smaller increments.
- Flexible for future upgrades (multi-language, voice chat, advanced analytics).

Explanation for This Project:

1. Requirements Gathering – Collect needs for:

- User authentication and account management (Supabase).
- Chatbot role definition form.
- Document upload system for knowledge base creation (PDF, DOCX, TXT).
- Embedding generation and storage in Supabase vector store.
- Real-time chatbot interface powered by Gemini API (RAG-enabled).

- Drag-and-drop chatbot preview and customization interface.
- JavaScript snippet generator for embedding chatbot in external websites.

2. System Design –

- Architecture integrating frontend (Vite), backend (Supabase Functions), vector storage, and Gemini

LLM API.

- Database schema for multi-tenant chatbot storage and embeddings.
- Data flow from user document → parsing → embedding generation → storage → retrieval → chatbot answer.
- API design for chat requests, embedding retrieval, and widget serving.

3. Implementation –

- Develop modules in sequence:

(1) Authentication + dashboard →

(2) Role definition + basic chatbot →

(2) Document upload + embeddings →

(2) Chatbot preview customization →

(2) JavaScript snippet generation →

(2) Advanced features (analytics, multi-language).

4. Testing –

- Functional testing for authentication, file uploads, and chatbot responses.
- Validation of UI animations, drag-and-drop customization, and widget embedding.
- Performance testing for embedding search and LLM response times.
- Security testing for data privacy and injection vulnerabilities.

5. Deployment –

- Launch early version with role-based chatbot and Gemini integration (no document upload).
- Add document-based chatbot with embeddings in second release.
- Release customization and embedding features in later increments.

6. Maintenance –

- Update chatbot UI and customization features based on user feedback.
- Optimize embeddings and LLM prompt engineering for better accuracy.
- Add new integrations (voice chat, analytics dashboard).
- Monitor and enhance security, scalability, and performance.

Waterfall SDLC Model: