**Functional Requirements**

**1. Chatbot to Gather User Information**

* **Chat Interaction**: The system will have a chatbot that asks the user questions like "What type of website do you want?" (e-commerce, blog, portfolio) and gather details.
* **Understand User Input**: The chatbot will understand the user’s responses and create a website plan (e.g., product categories for an online store or blog sections).

**2. Template Options for Websites**

* **Pre-made Templates**: Offer ready-to-use templates for e-commerce, blogs, and portfolios.
* **Customizable Design**: Users can choose different design elements (e.g., headers, footers, color schemes) for their website.

**4. API for E-commerce, Blogs, and Portfolios**

* **E-commerce**: Handle products, categories, and checkout functions for a basic store.
* **Blog**: Create and manage posts, categories, and tags.
* **Portfolio**: Showcase projects and skills.

**5. AI Model (Keras) for Website Suggestions**

* **Template Suggestions**: Use a trained AI model to suggest website templates based on user input (e.g., an online store for clothing will get a fashion-oriented design).

**6. Customizing Websites with GrapesJS**

* **Drag-and-Drop Editor**: Integrate **GrapesJS** to allow users to easily edit their website by dragging and dropping elements like text, images, buttons, etc.
* **Live Preview**: Users can see their changes in real-time as they edit their website.

**7. Downloading the Website**

* **Source Code Download**: After editing, users can download the website as a ZIP file that contains all the HTML, CSS, and JS files for their site.
* **Editable Files**: The downloaded website should be fully editable by users later on if they want to modify it manually.

**8. Flask as the Backend**

* **Flask API**: Use **Flask** to manage communication between the chatbot, AI model, and the template engine. It will serve as the backend to process user input and generate the website.

**Non Functional Requirements**

**Performance**

1. The system **must be** able to respond to user requests within **2 seconds**.
2. Searches for Api data **must be** completed in **1 second** or less.
3. Templates and previews **must be** updated in **real-time** during editing.
4. The system **must be** able to run smoothly on **low-bandwidth** connections.

**Reliability**

1. The system **must be** able to handle temporary internet disruptions.
2. The system **must be** able to handle gradual degradation under heavy load.

**Security**

1. The system **must be** able to **require user login** to access template creation and editing features.
2. The system **must be** able to **prevent unauthorized users** from viewing or modifying templates.
3. The system **must be** able to **keep user sessions secure** by using Flask’s built-in session management.

**Usability**

1. The interface **must be** simple and intuitive for non-technical users.
2. Users **must be** able to preview their websites in **real-time** while editing.
3. The system **must be** mobile-friendly, ensuring easy use on all devices.
4. Users **must be** able to undo/redo changes during website editing.

**Interoperability**

1. The system **must be** able to export completed websites as **ZIP files**.
2. The system **must be** able to import content from **HTML/CSS files**.
3. The system **must be** able to integrate with third-party systems using an API.
4. The system **must be** flexible enough to support third-party plugins for additional functionality.

**Maintainability**

1. The system **must be** modular, allowing easy updates and feature additions.
2. The system **must be** well-documented for both developers and users.

**Scalability**

1. The system **must be** designed to accommodate increases in data storage without performance degradation.
2. The system **must be** scalable enough to support new features without major rewrites.
3. Specific components of the system **must be** scalable independently (e.g., the chatbot or API).
4. The system **must be** able to add new templates and design elements over time with minimal effort.

**Availability**

1. The system **must be** available **24/7** with minimal downtime (less than 1 hour per month).
2. The system **must be** accessible from any device with an internet connection.

**Regulatory**

1. The system **must** protect customer data privacy.
2. The system **must be** audited regularly for compliance.

**Manageability**

1. The system **must be** easy to **install** and configure for both technical and non-technical users.
2. The system **must be** easily configurable to adapt to new templates or features.
3. The system **must be** supported by clear **documentation** for administrators and developers.

**Environmental**

1. The system **must be** designed to minimize its **environmental impact** by optimizing energy usage.
2. The system **must be** scalable in a way that reduces **resource consumption** as it grows.
3. The system **must be** designed to run efficiently on **minimal hardware** or in low-power settings.

**Use Case: Create E-Commerce Website**

**Function Name:**

Create E-Commerce Website

**Primary Actor:**

Website Administrator/Developer

**Trigger:**

The administrator/developer decides to start the creation of a new e-commerce website.

**Goal:**

To develop and implement a fully functional e-commerce website that meets specified business requirements.

**Precondition:**

* The administrator/developer has the necessary credentials and permissions.
* Required tools and technologies are installed (e.g., web server).

**Postcondition:**

* An e-commerce website is developed, meeting specified business requirements.

**Basic Flow:**

1. **Setup Development Environment:**
   * The administrator/developer sets up the development environment and necessary tools.
2. **Define Website Requirements:**
   * The administrator/developer gathers and defines detailed requirements for the e-commerce website (e.g., features, design, user roles).
3. **Design Website Layout:**
   * The administrator/developer creates wireframes and design mockups based on the requirements.
4. **Test Website Functionality:**
   * The administrator/developer performs comprehensive testing to ensure the website’s functionality, performance, and security.
5. **Prepare for Launch:**
   * The administrator/developer prepares the website for launch, that it is properly configured.

**Alternate Flow:**

* **If Issues are Detected During Testing:**
  + The administrator/developer addresses issues and re-tests until all functional requirements are met before proceeding to launch preparation.

**Quality Requirements:**

* It should be optimized for performance, security, and usability.
* The design must be responsive and compatible with various devices and browsers.

**Exception:**

* **If Technical Issues Occur During Development:**
  + The administrator/developer should handle any technical issues by diagnosing the problem, resolving it, and updating the system documentation as necessary.

**Use Case: Create Blog Website**

**Function Name:**

Create Blog Website

**Primary Actor:**

Website Administrator/Developer

**Trigger:**

The administrator/developer decides to start the creation of a new blog website.

**Goal:**

To develop and implement a blog website that allows for content creation, management, and interaction.

**Precondition:**

* The administrator/developer has the necessary credentials and permissions.
* Required tools and technologies are installed (e.g., web server, database).

**Postcondition:**

* A blog website is developed, meeting specified requirements for content management and user interaction.

**Basic Flow:**

1. **Setup Development Environment:**
   * The administrator/developer sets up the development environment and necessary tools.
2. **Define Website Requirements:**
   * The administrator/developer gathers and defines detailed requirements for the blog website, including features like post creation, categories, comments, and user roles.
3. **Design Website Layout:**
   * The administrator/developer creates wireframes and design mockups based on the defined requirements.
4. **Develop Website Components:**
   * The administrator/developer develops core components including blog post creation/editing, category management, and comment functionality.
5. **Prepare for Launch:**
   * The administrator/developer prepares the website for launch by ensuring all content is ready and the site is properly configured.

**Alternate Flow:**

* **If Issues are Detected During Testing:**
  + The administrator/developer addresses issues and re-tests the website to ensure all functionalities meet the requirements before preparing for launch.

**Quality Requirements:**

* The website should support robust content management and user interaction.
* It should be optimized for performance, security, and usability.
* The design must be responsive and compatible with various devices and browsers.

**Exception:**

* **If Technical Issues Occur During Development:**
  + The administrator/developer should handle technical issues by diagnosing problems, resolving them, and updating system documentation as necessary

**Use Case: Create Portfolio Website**

**Function Name:**

Create Portfolio Website

**Primary Actor:**

Website Administrator/Developer

**Trigger:**

The administrator/developer decides to start the creation of a new portfolio website.

**Goal:**

To develop and implement a portfolio website that effectively showcases work, skills, and personal information.

**Precondition:**

* The administrator/developer has the necessary credentials and permissions.
* Required tools and technologies are installed (e.g., web server, content management system).

**Postcondition:**

* A fully functional portfolio website is developed, meeting specified requirements for showcasing work, skills, and personal information.

**Basic Flow:**

1. **Setup Development Environment:**
   * The administrator/developer sets up the development environment and necessary tools.
2. **Define Website Requirements:**
   * The administrator/developer gathers and defines detailed requirements for the portfolio website, including features like project showcases, resume sections, contact forms, and design preferences.
3. **Design Website Layout:**
   * The administrator/developer creates wireframes and design mockups based on the defined requirements.
4. **Prepare for Launch:**
   * The administrator/developer prepares the website for launch by ensuring all content is up-to-date and the site is properly configured.

**Alternate Flow:**

* **If Issues are Detected During Testing:**
  + The administrator/developer addresses issues and re-tests the website to ensure all functionalities meet the requirements before proceeding to launch preparation.

**Quality Requirements:**

* The website should effectively showcase work and skills in a visually appealing manner.
* It should be optimized for performance, security, and usability.
* The design must be responsive and compatible with various devices and browsers.

**Exception:**

* **If Technical Issues Occur During Development:**
  + The administrator/developer should handle technical issues by diagnosing problems, resolving them, and updating system documentation as necessary.

**Use Case: Create and Implement Chatbot API**

**Function Name:**

Create and Implement Chatbot API

**Primary Actor:**

API Developer

**Trigger:**

The developer decides to create and implement a chatbot API to provide automated conversational capabilities.

**Goal:**

To develop, implement a functional chatbot API that integrates with a machine learning model for natural language processing (NLP) to handle user interactions effectively.

**Precondition:**

* The developer has the necessary credentials and permissions.
* Required tools and technologies are installed (e.g., development environment, machine learning libraries like Keras, NLTK).

**Postcondition:**

* A chatbot API is developed, integrated with a trained NLP model, and ready to use to take hand in websites contents.

**Basic Flow:**

1. **Define Chatbot Requirements:**
   * The developer gathers and defines detailed requirements for the chatbot API, including functionalities, user interactions, and integration needs.
2. **Design Chatbot Architecture:**
   * The developer designs the architecture of the chatbot, including the API endpoints, interaction flows, and integration points with the NLP model.
3. **Develop NLP Model:**
   * The developer develops and trains the NLP model using Keras and NLTK to handle various conversational intents and responses.
4. **Implement API Endpoints:**
   * The developer implements the API endpoints required for the chatbot, including endpoints for sending messages, receiving responses.
5. **Integrate NLP Model with API:**
   * The developer integrates the trained NLP model with the API, ensuring that it can process user inputs and generate appropriate responses.

**Alternate Flow:**

* **If Issues are Detected During Testing:**
  + The developer addresses issues and re-tests the API until all functional requirements are met before proceeding to deployment.

**Quality Requirements:**

* The API should be optimized for performance, security, and scalability.
* The NLP model should provide accurate and relevant responses based on user inputs.

**Exception:**

* **If Technical Issues Occur During Development or Deployment:**
  + The developer should handle technical issues by diagnosing problems, resolving them, and updating system documentation as necessary.

**Use Case: Edit Templates with GrapesJS**

**Function Name:**

Edit Templates with GrapesJS

**Primary Actor:**

Web Developer

**Trigger:**

The developer decides to use GrapesJS to edit and customize website templates.

**Goal:**

To edit and customize website templates using GrapesJS, enabling visual and code-based modifications of HTML and CSS.

**Precondition:**

* The developer has GrapesJS and necessary dependencies installed.
* The template to be edited is available and accessible.

**Postcondition:**

* The website templates are updated and customized according to the developer's specifications using GrapesJS.

**Basic Flow:**

1. **Initialize GrapesJS Editor:**
   * The developer sets up and initializes the GrapesJS editor within the development environment.
2. **Load Template:**
   * The developer loads the existing website template into GrapesJS for editing.
3. **Edit Template:**
   * The developer utilizes GrapesJS’s drag-and-drop interface and code editor to:
     + Add, remove, or rearrange components (e.g., text, images, buttons).
     + Modify styles and layouts using visual editing tools.
     + Directly edit HTML/CSS code as needed.
4. **Preview Changes:**
   * The developer previews the changes in real-time to verify the look and functionality of the edits.
5. **Save Edits:**
   * The developer saves the updated template. GrapesJS allows exporting of the modified HTML and CSS code.
6. **Integrate Changes:**
   * The developer integrates the saved changes into the main project, ensuring the updated template is reflected in the website.
7. **Test Template:**
   * The developer tests the updated template across different devices and browsers to ensure compatibility and responsiveness.
8. **Deploy:**
   * After testing, the developer deploys the updated template to the production environment.

**Alternate Flow:**

* **If Issues are Detected During Editing:**
  + The developer makes necessary adjustments in GrapesJS, re-tests the changes, and re-saves until all issues are resolved.

**Quality Requirements:**

* The template should be editable with an intuitive drag-and-drop interface.
* Real-time previews must accurately reflect changes.
* The final template should be responsive and compatible with various devices and browsers.
* Documentation should be provided for the editing process and any troubleshooting.

**Exception:**

* **If Technical Issues Occur During Editing or Saving:**
  + The developer should troubleshoot the issues, resolve them, and update any relevant documentation.

**Use Case: Edit Templates with GrapesJS**

**Function Name:**

Edit Templates with GrapesJS

**Primary Actor:**

Web Developer

**Trigger:**

The developer decides to use GrapesJS to edit and customize website templates.

**Goal:**

To edit and customize website templates using GrapesJS, enabling visual and code-based modifications of HTML and CSS.

**Precondition:**

* The developer has GrapesJS and necessary dependencies installed.
* The template to be edited is available and accessible.

**Postcondition:**

* The website templates are updated and customized according to the developer's specifications using GrapesJS.

**Basic Flow:**

1. **Initialize GrapesJS Editor:**
   * The developer sets up and initializes the GrapesJS editor within the development environment.
2. **Load Template:**
   * The developer loads the existing website template into GrapesJS for editing.
3. **Edit Template:**
   * The developer utilizes GrapesJS’s drag-and-drop interface and code editor to:
     + Add, remove, or rearrange components (e.g., text, images, buttons).
     + Modify styles and layouts using visual editing tools.
     + Directly edit HTML/CSS code as needed.
4. **Preview Changes:**
   * The developer previews the changes in real-time to verify the look and functionality of the edits.
5. **Save Edits:**
   * The developer saves the updated template. GrapesJS allows exporting of the modified HTML and CSS code.
6. **Integrate Changes:**
   * The developer integrates the saved changes into the main project, ensuring the updated template is reflected in the website.

**Alternate Flow:**

* **If Issues are Detected During Editing:**
  + The developer makes necessary adjustments in GrapesJS, re-tests the changes, and re-saves until all issues are resolved.

**Quality Requirements:**

* The template should be editable with an intuitive drag-and-drop interface.
* Real-time previews must accurately reflect changes.
* The final template should be responsive and compatible with various devices and browsers.

**Exception:**

* **If Technical Issues Occur During Editing or Saving:**
  + The developer should troubleshoot the issues, resolve them, and update any relevant documentation.

**Use Case: Download Customized Templates**

**Function Name:**

Download Customized Templates

**Primary Actor:**

Web Developer

**Trigger:**

The developer has customized a template and wishes to download it.

**Goal:**

To successfully download the customized template from GrapesJS in a format suitable for use or deployment.

**Precondition:**

* The developer has customized the template using GrapesJS.
* GrapesJS is properly configured and operational within the development environment.

**Postcondition:**

* The customized template is downloaded successfully in the desired format (e.g., HTML, CSS, and assets).

**Basic Flow:**

1. **Complete Customization:**
   * The developer finishes customizing the template using GrapesJS.
2. **Access Download Option:**
   * The developer navigates to the download option or menu within the GrapesJS interface.

**Alternate Flow:**

* **If Download Fails:**
  + The developer reattempts the download. If issues persist, troubleshooting steps are taken to resolve the problem, and the process is reinitiated.

**Quality Requirements:**

* The download process should be straightforward and accessible from the GrapesJS interface.
* The downloaded file should accurately reflect all customizations made.
* The format should be compatible with the intended use or deployment environment.

**Exception:**

* **If Technical Issues Occur During Download:**
  + The developer should diagnose and resolve technical issues, ensuring that the system provides clear error messages and support for troubleshooting.