

**SCHOOL OF COMPUTER SCIENCE AND APPLICATIONS**

A Project Report

On

**Multi-Language Text-to-Image Generator** **using AWS**

Submitted in Partial fulfillment of the requirements for the award of the Degree of

Bachelor of Science (Cloud computing and bid data)

Submitted by

**Bhavana LN**

**R22DB008**

Under the guidance of

**Dr. Devi A**

Internal Guide

**January 2025**

Rukmini Knowledge Park, Kattigenahalli, Yelahanka, Bengaluru-560064

[www.reva.edu.in](http://www.reva.edu.in)



**SCHOOL OF COMPUTER SCIENCE AND APPLICATIONS**

# CERTIFICATE

The project work titled **Multi-language** **Text**-**to image generator using AWS** is beingcarried out under our guidance by **Bhavana L N -R22DB008**, Bonafide students of REVA University, and is submitting the project report in partial fulfillment, for the award of **Bachelor of Science (Cloud computing and big data) in Computer Science** during the academic year **2024–25**. The project report has been approved, as it satisfies the academic requirements with respect to the Project Work prescribed for the aforementioned Degree.

**Signature with date Signature with date Signature with date**

**Dr. G. Sasikala**

**Head of the Department**

**BSc**

**External Guide**

**Dr. Devi A**

**Internal Guide**

**Signature with Date**

**­­­­­­­­­­­­­­­­­­­**

**Dr. C K Lokesh**

**Director**

**External Examiner**

**Name of the Examiner with** **Affiliation** **Signature with Date**

**1.**

**2.**

**DECLARATION**

I, Bhavana L N- R22DB008 pursuing our **Bachelor of Science (Cloud computing and big data)** offered by School of Computer Science and Applications, REVA University, declare that this Project titled - **“Text-to-image generator using AWS”** is the result of the Project Work done by me under the supervision of **Dr. Devi** at REVA University.

I am submitting this Project Work in partial fulfillment of the requirements for the award of the degree of **Bachelor of Science (Cloud computing and big data)** by REVA University, Bengaluru, during the Academic Year 2023-24.

I further declare that this Project Report or any part of it has not been submitted for the award of any other Degree / Diploma of this University or any other University/ Institution.

*Bhavana L N*

*Signed by me on:*

*Certified that this project work submitted* ***Bhavana L N- R22DB008*** *has been carried out under our guidance and the declaration made by the candidate is true to the best of my knowledge.*

*Signature of Internal Guide Signature of External Guide*

*Date :……….. Date :………..*

*Signature of Director of the School*

*Date :………..*

*Official Seal of the School*

**ACKNOWLEDGEMENT**

I hereby acknowledge all those, under whose support and encouragement, I have been able to fulfil all my academic commitments successfully. In this regard, I take this opportunity to express my deep sense of gratitude and sincere thanks to School of Computer Science and Applications which has always been a tremendous source of guidance.

I express my sincere gratitude to **Dr. P. SHYAMA RAJU**, Honourable Chancellor, REVA University, and Bengaluru for providing us the state-of-the-art facilities.

I am thankful to **Dr. N RAMESH**, Vice Chancellor (i/c), REVA University, **Dr K S NARAYANASWAMY**, Registrar (i/c), REVA University and **Dr. SANJAY CHITNIS**, Pro-Vice Chancellor, REVA University, Bengaluru for their support and encouragement.

I take this opportunity to express my heartfelt thanks to **Dr. C K Lokesh**, Director, School of CSA, REVA University and my sincere thanks to Dr. G. SASIKALA Associate Professor, Head of the Department for BSc Program, School of CSA, REVA University, whose encouragement and best wishes provided impetus for the Project Work carried out.

Also, my sincere gratitude goes to my internal guide, Dr. Devi A, for the valuable suggestion and constant encouragement towards the completion of this project.

Last, but not the least, I thank my parents for their incredible support and encouragement throughout.

**ABSTRACT**

The “**Multi-Language Text-to-Image Generator using AWS”** is an innovative project that bridges language processing and image generation. This system allows users to input text prompts in multiple languages and generates corresponding images based on the textual descriptions.

The project leverages Flask as the backend framework for handling user interactions and input processing, along with Python libraries for text-to-image generation. The images generated are stored securely in an AWS S3 bucket, ensuring scalability and data persistence. The entire application is deployed on AWS, utilizing its robust infrastructure to provide high availability, speed, and reliability.

This application caters to a global audience by supporting multiple languages, making it accessible and inclusive. It combines the power of natural language processing, image generation, and cloud services to deliver a seamless and efficient user experience. The project’s implementation showcases the integration of modern technologies to solve real-world problems in creative content generation.

**TABLE OF CONTENTS**

**CHAPTERS PAGE NO**

**1. INTRODUCTION 8 -13**

1.1 INTRODUCTION TO PROJECT

- STATEMENT OF THE PROBLEM

- BRIEF DESCRIPTION OF THE PROJECT

- SOFTWARE AND HARDWARE SPECIFICATION

1.2 FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

**2. LITERATURE SURVEY 14-16**

**3. SYSTEM ANALYSIS 17-21**

3.1 EXISTING SYSTEM

3.2 LIMITATIONS OF THE EXISTING SYSTEM

3.3 PROPOSED SYSTEM

3.4 ADVANTAGES OF THE PROPOSED SYSTEM

3.5 FEASIBILITY STUDY

- TECHNICAL FEASIBILITY

- ECONOMICAL FEASIBILITY

- OPERATIONAL FEASIBILITY

4. **SYSTEM DESIGN AND DEVELOPMENT 22-40**

4.1 HIGH LEVEL DESIGN (ARCHITECTURAL)

4.2 LOW LEVEL DESIGN

4.3 ENTITY-RELATIONSHIP DIAGRAM

4.4 DATAFLOW DIAGRAM

4.5 USE CASE DIAGRAM

4.6 SEQUENCE DIAGRAM

4.7 CLASS DIAGRAM

4.8 ACTIVITY DIAGRAM

4.9MODULE DESCRIPTION

**5. CODING 41-52**

5.1 PSEUDO CODE

**6. SOFTWARE TESTING (Test cases) 53-56**

**7. CONCLUSION AND SCOPE FOR FUTURE ENCHANCEMENT 57-58**

**BIBILOGRAPHY 59**

**APPENDIX 60-70**

1. **SNAPSHOTS**

**1.Introduction**

* 1. **Introduction to Project**

The **Multi-Language Text-to-Image Generator** is a cutting-edge application that combines the power of natural language processing, image generation, and cloud computing to provide an innovative solution for creating visual content from textual descriptions in multiple languages. The project is designed to cater to a global audience, addressing the need for inclusivity and accessibility in creative content generation.

The system operates through a simple, user-friendly web application built with Flask. Users can input text prompts in various languages, and the application processes these inputs to generate corresponding images. The generated images are stored securely in AWS S3, leveraging cloud scalability and durability. The entire application is hosted on AWS, ensuring high availability and robust performance.

**Statement of the Problem**

In a world increasingly driven by digital innovation, there is a rising demand for tools that can seamlessly transform textual content into visual formats. This demand spans industries like marketing, education, gaming, and content creation. While existing text-to-image generation tools are making strides in this domain, they often have significant limitations:

**Language Constraints:** Many tools are designed to process text input in only one or a few languages, excluding a vast segment of the global population.

**Technical Complexity:** Non-technical users face challenges in interacting with such systems, as they require complex setups or programming knowledge.

**Scalability and Accessibility:** The lack of robust infrastructure in some solutions results in slower processing speeds, limited scalability, and reduced accessibility.

This project aims to address these challenges by creating a multi-language text-to-image generation tool that is easy to use, scalable, and efficient, ensuring inclusivity and accessibility for a global user base.

**Brief Description of the Project**

The Multi-Language Text-to-Image Generator is an advanced system designed to convert textual descriptions into images in real time, supporting input in multiple languages. Built using Flask as the backend framework, the system provides a simple and user-friendly web interface for users to input text prompts.

When a user submits text, the system processes it using Python-based text-to-image generation models, which leverage cutting-edge machine learning techniques. The resulting images are stored in an AWS S3 bucket for secure and scalable storage. The application is deployed on AWS infrastructure, ensuring high availability, low latency, and seamless scalability to handle a growing number of users.

The integration of multi-language support ensures that the application can cater to a diverse audience, enabling global accessibility. The system’s architecture emphasizes scalability, reliability, and ease of use, making it a powerful tool for generating visual content across languages and cultures.

**Software and Hardware Specification**

**Software Requirements:**

* **Programming Languages:** Python, HTML
* **Framework:** Flask
* **Cloud Services:** AWS :-S3 for storage, EC2 for deployment
* **Libraries:** Text-to-image generation libraries Natural Language Processing libraries
* **Operating System:** Linux/Windows/MacOS

**Hardware Requirements:**

* **Processor:** Minimum quad-core processor
* **RAM:** 8 GB or higher
* **Storage:** 50 GB or more
* **Cloud Infrastructure:** AWS account with S3 bucket and EC2 instance configured
* **GPU (optional):** Recommended for faster image generation, such as NVIDIA CUDA-compatible GPUs.

**1.2Functional requirements.**

**User Input Handling:**

* Accept textual descriptions from users in various languages through a web-based interface.
* Provide support for multi-language input, including special characters and non-Latin scripts.

**Language Processing:**

* Translate non-English text into a format understandable by the text-to-image generation model, using NLP libraries or APIs (e.g., Google Translate API).
* Handle synonyms, grammar variations, and idiomatic expressions in textual inputs to ensure accurate image generation.

**Image Generation:**

* Process user input to generate an image using advanced text-to-image generation models (e.g., Stable Diffusion).
* Allow for customization options, such as adjusting image dimensions, styles, or themes if required.

**Image Storage:**

* Store generated images in an AWS S3 bucket with appropriate folder structures for user-specific or session-based organization.
* Enable retrieval and display of stored images on the web interface.

**Web Application Features:**

* Provide a user-friendly interface built with Flask and HTML for text submission and image display.
* Include navigation features like home, upload, and view gallery.
* Display error messages for invalid inputs or system errors.

**User Feedback:**

* Allow users to download generated images.
* Provide options for user feedback on image quality and application usability.

**System Administration:**

* Include a logging mechanism to track system performance, user interactions, and error reports.
* Enable admin-level access to manage user sessions, logs, and storage.

**Non-functional requirements.**

**Performance:**

* Ensure real-time or near-real-time image generation, with a maximum processing time of 5 seconds per request under normal conditions.

**Scalability**:

* Design the system architecture to accommodate increased user traffic and storage demands by leveraging AWS’s auto-scaling features.

**Reliability:**

* Achieve 99.9% uptime for the web application, ensuring continuous availability for users.

**Security:**

* Utilize AWS IAM policies to ensure secure access to S3 buckets and EC2 instances.
* Encrypt stored images in S3 to protect sensitive data.

**Usability:**

* Design the web interface with intuitive navigation and clear instructions for users.

**Accessibility:**

* Ensure the application meets web accessibility standards (e.g., WCAG 2.1) to cater to users with disabilities.

**Portability:**

* Deploy the application on AWS infrastructure but maintain portability to migrate to other cloud platforms if needed.

**Compliance:**

* Ensure compliance with data protection regulations like GDPR for international users.

**2.Literature Survey**

The literature survey examines existing technologies, methodologies, and systems related to text-to-image generation, multi-language support, and cloud-based application development. It provides insights into the evolution of the field, identifies gaps in existing solutions, and highlights the innovations introduced by this project.

1. **Text-to-Image Generation**

Text-to-image generation has emerged as a significant area of research in artificial intelligence, particularly in the domain of generative models. Some key contributions include:

**Diffusion Models:** Recent advancements like DALL-E, Stable Diffusion, and Mid Journey leverage diffusion models to generate high-quality, semantically accurate images. These models use transformer-based architectures to encode textual prompts into visual representations effectively.

**Challenges in Text-to-Image Systems:**

* Achieving semantic alignment between text and generated images.
* Processing complex or abstract text inputs.
* Balancing image quality with computational efficiency.

2.**Multi-Language Support in NLP**

Handling multi-language inputs involves several challenges, including translation accuracy, syntactic differences, and semantic nuances. Key advancements include:

* **Google Translate API:** Provides robust multi-language translation capabilities, enabling applications to process text in over 100 languages.
* **Transformers (e.g., BERT, GPT):** Transformer-based architectures have significantly improved the understanding and generation of text across multiple languages.
* **Multilingual NLP Models:** Models like mBERT and XLM-RoBERTa are designed for processing and understanding text in multiple languages with high accuracy.

3**.Cloud-Based Solutions**

Cloud platforms like AWS provide scalable and reliable infrastructure for deploying machine learning applications. Key AWS services relevant to this project include:

* **Amazon S3:** A highly scalable storage service ideal for storing generated images securely.
* **Amazon EC2:** Provides virtual servers to host applications and process user requests.
* **AWS IAM:** Ensures secure and granular access control to resources.
* **Amazon Translate:** Can be integrated for multi-language text processing if needed.

4.**Existing Solutions and Gaps**

Several text-to-image systems exist, but they often have the following limitations:

* Lack of support for multiple languages.
* Dependency on local hardware, limiting scalability and performance.
* Complex user interfaces that make them inaccessible to non-technical users.
* Limited storage and retrieval capabilities for generated images.

5. **Innovations in This Project**

This project addresses these gaps by:

* Supporting multi-language text inputs through robust NLP techniques.
* Leveraging state-of-the-art text-to-image models for accurate and high-quality image generation.
* Using AWS cloud services for scalability, reliability, and secure storage.
* Providing a simple, intuitive web interface for users, making the system accessible to a non-technical global audience.

**3.System Analysis**

**3.1 Existing System**

Currently, text-to-image generation systems are either limited in language support, scalability, or both. Many systems are single-language and focused on English, limiting their global reach. These systems also often rely on local hardware resources, which can hinder scalability and performance, especially for large numbers of concurrent users. Some notable existing systems include:

* AttnGAN (Attention Generative Adversarial Networks): A widely recognized model that generates images from text using attention mechanisms, but it mainly supports English text and often requires significant computational resources.
* DALL-E: An advanced model developed by OpenAI capable of generating detailed images from text prompts, but it may not efficiently handle multi-language inputs and relies on high-performance infrastructure.
* Image Generation APIs: Platforms like DeepAI and RunwayML offer image generation through simple API calls, but they do not provide built-in multi-language support or seamless cloud storage solutions.

**3.2 Limitations of the Existing System**

The main limitations of the existing systems are:

1. **Lack of Multi-Language Support:** Most current systems are designed for English input and may struggle with non-Latin scripts or languages with complex sentence structures.
2. **Scalability Issues:** Many existing systems are not designed to scale efficiently, especially when handling a large number of simultaneous users or requests.
3. **High Computational Costs:** High-performance models like DALL-E or GAN-based systems require significant hardware resources, such as GPUs, which may not be feasible for all users, especially in a commercial setting.
4. **Limited Accessibility:** Many text-to-image generation systems require technical knowledge to set up and use, creating a barrier for non-technical users.
5. **Inconsistent Image Quality:** While some systems generate high-quality images, others may produce low-resolution or poorly aligned images based on the input text.

These limitations highlight the need for a more scalable, accessible, and inclusive text-to-image generation system.

**3.3 Proposed System**

The proposed system is designed to overcome the limitations of the existing systems by providing a multi-language, scalable, and user-friendly platform for generating images from text descriptions. Key features of the proposed system include:

1. **Multi-Language Support:** The system will handle text input in multiple languages, using advanced NLP techniques to process and translate non-English text into a format suitable for image generation.
2. **Cloud-Based Infrastructure:** By leveraging AWS cloud services (e.g., EC2, S3), the system ensures scalability, security, and easy access from anywhere, without the need for local hardware.
3. **Text-to-Image Generation:** Utilizing advanced machine learning models (e.g., Stable Diffusion), the system will generate high-quality images based on user inputs.
4. **User-Friendly Interface:** The system will have an intuitive web interface built with Flask and HTML, ensuring that users, regardless of their technical expertise, can generate images easily.
5. **Secure and Scalable Image Storage:** Generated images will be stored in AWS S3 buckets, offering high availability, secure access, and scalability for growing storage needs.

**3.4 Advantages of the Proposed System**

The proposed system provides several advantages over existing systems:

1. **Global Accessibility:** Multi-language support ensures that users from different linguistic backgrounds can easily generate images without language barriers.
2. **Scalability and Reliability:** By leveraging AWS infrastructure, the system can scale automatically to handle increased traffic and demand, ensuring consistent performance under high loads.
3. **Cost-Effectiveness:** The system reduces the need for high-performance local hardware, as it relies on cloud resources, lowering the upfront costs for users.
4. **Security:** AWS S3 ensures that all generated images are securely stored, with access control policies in place to protect user data.
5. **Ease of Use:** With an intuitive Flask-based web interface, users without technical expertise can easily interact with the system and generate images from text prompts.
6. **Faster Image Generation:** The integration of state-of-the-art image generation models allows for faster, more accurate image creation.

**3.3 Feasibility Study**

A feasibility study is essential to assess whether the proposed system is practical and achievable. It involves evaluating the technical, economic, and operational aspects of the system.

**Technical Feasibility:**

The technical feasibility of the proposed system is high, as it leverages existing technologies and frameworks that are well-documented and widely used in the industry:

* **Cloud Infrastructure:** AWS provides reliable and scalable cloud services, which are essential for hosting the system and handling large amounts of user data. EC2 instances can be easily scaled to accommodate increased traffic, and S3 offers secure storage for the generated images.
* **Machine Learning Models:** The text-to-image generation models (e.g., Stable Diffusion) have been extensively tested and proven to generate high-quality images, and integrating them with the Flask backend is straightforward.
* **NLP for Multi-Language Support:** The use of NLP models and translation services, such as Google Translate or AWS Translate, ensures accurate translation and processing of text inputs in multiple languages.
* **Web Development:** Flask is a lightweight web framework ideal for building scalable web applications. It is easy to integrate with Python-based machine learning models and APIs for generating images.

**Economical Feasibility:**

The economic feasibility of the proposed system is also favorable:

* **Cost of Cloud Resources:** Using AWS services such as EC2 and S3 provides a cost-effective solution, as costs scale with usage. The system only incurs costs based on usage (pay-as-you-go model), which reduces upfront investment.
* **Reduced Hardware Costs:** Since the system relies on cloud computing resources rather than requiring local GPU hardware, users and developers do not need to invest in expensive hardware setups.
* **Revenue Potential:** The system could generate revenue through paid API access, premium features, or subscription-based models. By supporting global users and offering easy integration with other services, it has the potential to capture a large audience, especially in industries like marketing, content creation, and education.

**Operational Feasibility:**

Operational feasibility refers to whether the proposed system can be effectively used and maintained:

**Ease of Use:** The web interface is designed to be intuitive and user-friendly, which ensures that non-technical users can generate images with ease. Minimal training or technical support will be required.

**Maintenance:** The system is built on well-established technologies (Flask, AWS, Python), making it easy to maintain and update. Regular updates to machine learning models and system optimizations can be handled without significant disruptions.

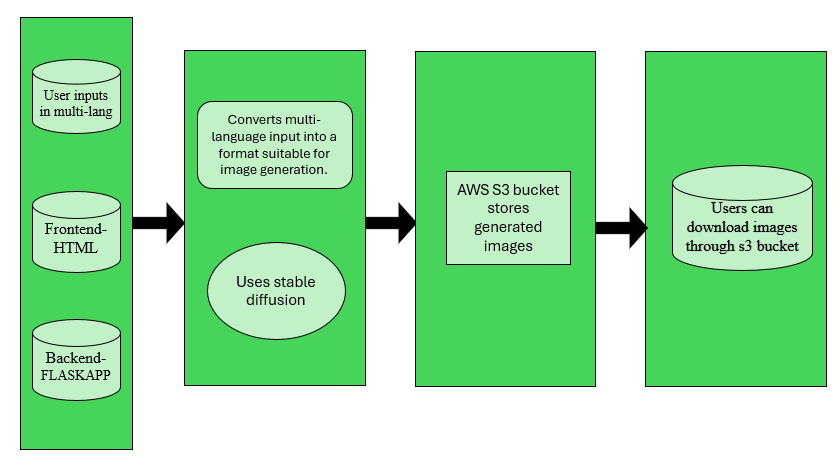
**Support:** The system relies on robust cloud infrastructure, meaning that issues related to downtime, data loss, or scaling can be managed efficiently through AWS support and monitoring services.

**Global Accessibility:** Since the system is cloud-based, it is accessible from anywhere in the world, which improves its operational reach.

Overall, the operational feasibility of the system is high, as it aligns with industry standards and provides a solution that can be easily maintained and operated on a global scale.

**4. System design and development**

**4.1High level diagram (architecture)**

****

The high-level architecture outlines the system's overall structure, describing its key components, interactions, and the flow of data. This architecture is designed for scalability, efficiency, and ease of use, leveraging cloud infrastructure and advanced machine learning techniques. Below is the detailed description of the high-level architecture

* Users input text in multiple languages via a browser-based UI.
* Technologies: HTML, CSS, JavaScript.
* Key Actions: Accept text input, validate it, and send it to the backend.
* Backend Flask app handles API requests.
* Integrates with AWS API Gateway (optional) for routing.
* Key Actions: Validate requests, interact with the processing layer, and return responses.
* Python logic processes the text.
* Converts multi-language input into a format suitable for image generation.
* AWS S3 bucket stores generated images.
* Images are tagged for easy retrieval.
* Flask app fetches images from S3 and sends them to the user.

**4.2 Low-level design:**

**A screenshot of a computer

Description automatically generated**

* **Frontend Layer:**
* User Interface: HTML, CSS, and JavaScript.
* Form Input: Fields to accept text in multiple languages.
* Validation: JavaScript ensures valid input (e.g., text length, prohibited characters).
* **Backend Layer:**
* Framework: Flask (Python-based backend framework).
* Routes: Define specific endpoints (e.g., /generate-image, /status).
* Controller Functions: Handle incoming requests and direct them to the processing layer.
* **Processing Layer:**
* **Language Processing:**
  + Detects the language using libraries (e.g., Google Translate API or Hugging Face Transformers).
  + Translates non-English text to English if necessary.
* **Image Generation:**
* Uses text-to-image conversion pipelines with fine-tuning for custom styles.
* GPU acceleration if deployed on AWS EC2 or Lambda with GPU support.
* **Storage Layer:**
* **AWS S3:**
  + Bucket configuration for storing images.
  + Tagging and folder structure (e.g., /userID/images/).
* **Pre-Signed URLs:**
  + Generate time-limited URLs for secure access to images.
* **Output Delivery Layer:**
* **Image Retrieval**:
  + Backend fetches the image from AWS S3.
* Response:
  + Sends the image URL or binary data back to the frontend.

**4.3ENTITY-RELATIONSHIP DIAGRAM**

Image

Language

Text Input

Input ID(PK)

Language ID (FK)

Text Description

Submission Timestamp

Language

Language ID(PK)

Language name

Generated name

Image IDPK)

Input ID (FK)

Image URL

Generated Timestamp

AWS\_S3\_Bucket

Bucket

Model Processing

Bucket ID(PK)

Image ID(FK)

Storage URL

File size

Uploaded timestamp

Process ID(PK)

Input ID(FK)

Model name

Processing time

Output status

* **Text Input**
* Primary Key: Input ID
* Attributes:
  + Language ID (Foreign Key referencing Language. Language ID)
  + Text Description (Input text from the user)
  + Submission Timestamp (Timestamp of when the text was submitted)
  + Language (Detected or selected language)
* **Language**
* Primary Key: Language ID
* Attributes:
  + Language Name (Name of the language, e.g., English, Spanish)
  + Generated Image (Boolean/Flag to indicate image generation status)
* **Image**
* Primary Key: Image ID
* Attributes:
  + Input ID (Foreign Key referencing Text Input. Input ID)
  + Image URL (URL of the generated image in S3)
  + Generation Timestamp (Timestamp when the image was generated)
  + AWS\_S3\_Bucket (Name or ID of the S3 bucket where the image is stored)
* **Bucket**
* Primary Key: Bucket ID
* Attributes:
  + Image ID (Foreign Key referencing Image- Image ID)
  + Storage URL (Full URL or path to the bucket storage)
  + File Size (Size of the image file in bytes)
  + Upload Timestamp (Timestamp of when the image was uploaded)
* **Model Processing**
* Primary Key: Process ID
* Attributes:
  + Input ID (Foreign Key referencing Text Input. Input ID)
  + Model Name (Name of the AI model used for image generation)
  + Processing Time (Time taken to process the input and generate the image)
  + Output Status (Status of the output, e.g., Success, Failed)
* **Relationships**

1. Text Input to Language:
   * Relationship: One-to-Many
   * Description: A language can have multiple text inputs, but each text input belongs to one language.
2. Tex Input to Image:
   * Relationship: One-to-Many
   * Description: A text input can generate multiple images, but each image is associated with one text input.
3. Image to Bucket:
   * Relationship: One-to-One
   * Description: Each generated image is stored in one bucket.
4. Text Input to Model Processing:
   * Relationship: One-to-Many
   * Description: A single text input can undergo multiple processing steps (e.g., retries or alternative models).

**4.4 Data flow diagram**

**A diagram of a software process

Description automatically generated with medium confidence**

* **From User to Frontend**:
* Input: Text data.
* Validation: Ensure the text is in the correct format.
* **From Frontend to Backend**:
* API call: POST /generate-image.
* **Backend to Processing Layer**:
* Input: Text in JSON format.
* Output: Generated image file.
* **Processing Layer to Storage**:
* Upload: Store image in AWS S3.
* **Storage to Backend**:
* Pre-signed URL for image retrieval.
* **Backend to Frontend**:
* Output: Image URL or binary data

**4.5 Use case diagram:**

**A diagram of a software application

Description automatically generated**

* **Actors:**
* User (End User): Interacts with the web application to input text and generate images.
* AWS (System): Responsible for processing text inputs, generating images, and storing them in an S3 bucket.
* Admin: Manages the application, user accounts, and settings.
* **Description:**
* Generate Image: The user inputs text in various languages, and the system generates an image.
* View Image: The user can view the generated image.
* Download Image: The user can download the generated image.
* Upload Image: Admin uploads images to the system or S3 bucket for management.
* Manage User Accounts: Admin manages user accounts, e.g., creating, deleting, or updating user information.
* Access S3 Bucket: The system accesses the S3 bucket to store or retrieve images.
* Switch Language: The user can switch between multiple languages for input.

**4.6Sequence diagram:**

**A diagram of a computer program

Description automatically generated**

* **Participants:**
* User: The person interacting with the application.
* Frontend (Browser): The web interface running on the user’s device.
* Backend (Flask on AWS EC2): Handles requests, text-to-image generation, and S3 interactions.
* AWS S3: Stores the generated images and provides a URL for access.
* **Description:**
* **User Inputs Text:**
* The user enters a text prompt and selects a language in the frontend UI.
* The frontend sends this data to the Flask backend on AWS EC2 via an HTTP POST request.
* **Backend Validates Input**:
* The Flask backend receives the request.
* It validates the text input (e.g., checking for empty or unsupported inputs).
* **Backend Generates Image:**
* The backend processes the text input and uses a locally hosted machine learning model to generate an image.
* **Backend Uploads Image to S3**:
* Once the image is generated, the backend uploads it to an AWS S3 bucket.
* S3 returns a public or pre-signed URL for the uploaded image.
* **Backend Returns Image URL:**
* The Flask backend sends the image URL as a response to the frontend.
* **Frontend Displays Image:**
* The frontend receives the image URL and displays the generated image to the user.
* Optionally, the user can download the image via the provided link.

**4.7Class diagram:**

**A diagram of a software

Description automatically generated**

* **User Class:**
* Attributes:
  + user\_id: Unique identifier for the user.
  + username: The username of the user.
  + email: The email of the user.
* Methods:
  + submit\_text\_input(): Submits the text input to the application.
  + view\_image(): Views the generated image.
  + download\_image(): Downloads the image.
* **TextInput Class:**
* Attributes:
  + text: The text or prompt provided by the user.
  + language: The language in which the text is written.
* Methods:
  + validate\_input(): Validates the text input for any errors or unsupported characters.
* **Image Class:**
* Attributes:
  + image\_id: Unique identifier for the image.
  + image\_url: URL where the image is stored.
  + image\_format: Format of the generated image (e.g., PNG, JPEG).
* Methods:
  + generate\_image(): Generates the image based on the input text.
  + display\_image(): Displays the image to the user.
  + store\_image(): Stores the generated image in the S3 bucket.
* **ImageGenerator Class:**
* Attributes:
  + generator\_id: Unique identifier for the image generator.
  + model: Model used for image generation.
* Methods:
  + process\_text\_input(): Processes the user’s text input to generate an image.
  + generate\_from\_model(): Uses an AI model to generate the image based on the input.
* **S3Storage Class:**
* Attributes:
  + bucket\_name: Name of the S3 bucket where images are stored.
  + image\_key: The key or path to the image in the bucket.
* Methods:
  + upload\_image(): Uploads the image to the S3 bucket.
  + retrieve\_image(): Retrieves the image from the S3 bucket.
* **WebApplication (Flask) Class:**
* Attributes:
  + app: Flask app instance.
* Methods:
  + receive\_input(): Receives text input from the user.
  + send\_response(): Sends the generated image URL back to the user.

**4.8 Activity diagram:**

Submit text

Valid input

**YES NO**

Display error message

Generate image

Re-enter text

Store image in s3

Return image URL to user

View/Download image

* **Start**: The process begins with the user interacting with the system.
* **Submit Text Input**: The user provides a text prompt to the application.
* **Decision**: If the input is valid, the image generation process starts. If invalid, an error message is shown, and the user can try again.
* **Generate Image**: The application sends the input to the image generation service (e.g., AWS Lambda).
* **Store Image in S3**: The generated image is uploaded to the S3 bucket for storage.
* **Return Image URL to User**: The application fetches the image URL from S3 and returns it to the user.
* **User Views/Downloads Image**: The user can view or download the image.
* **End**: The process concludes.

**4.9 Module description:**

1. **User Interface Module**

* **Purpose**: Provides a user-friendly frontend for users to interact with the system.
* **Key Features**:
  + Text input field for user-provided descriptions.
  + Dropdown menu for selecting or auto-detecting the input language.
  + Submit button to trigger the image generation process.
  + Displays the generated image along with options to download or share it.
* **Technologies**:
  + HTML/CSS for layout and styling.

1. **Backend API Module**

* **Purpose**: Acts as the intermediary between the frontend and the processing engine.
* **Key Features**:
  + Receives user input from the frontend.
  + Validates and processes input data.
  + Routes requests to the machine learning module for image generation.
  + Sends the generated image and metadata back to the frontend.
* **Technologies**:
  + Flask (Python frameworks for API development).

1. **Text Processing Module**

* **Purpose**: Prepares user input for the image generation process.
* **Key Features**:
  + **Language Detection**: Automatically detects the language of the input text.
  + **Translation**: Converts non-English text into English using a service like AWS Translate or Google Translate.

1. **Cloud Integration Module**

* **Purpose**: Manages cloud storage and compute resources.
* **Key Features**:
  + **AWS S3**:
    - Stores generated images securely.
    - Organizes images with metadata tagging for efficient retrieval.
  + **AWS EC2**:
    - Hosts the machine learning models.
    - Ensures scalability with auto-scaling configurations.
* **Technologies**:
  + AWS SDK (Boto3 for Python).

1. **Security and Authentication Module**

* **Purpose**: Protects user data and ensures secure access to the system.
* **Key Features**:
  + **User Authentication**: Uses AWS Cognito for user sign-in and token-based authorization.
  + **Data Encryption**:
    - Encrypts data at rest (in AWS S3) using AWS KMS.
    - Secures data in transit using HTTPS.
  + **Role-Based Access Control (RBAC)**: Restricts access based on user roles.
* **Technologies**:
  + AWS IAM (Identity and Access Management).

**5. Coding**

from flask import Flask, request, render\_template\_string

import requests

import boto3

from googletrans import Translator

import os

from huggingface\_hub import hf\_api

app = Flask(\_\_name\_\_)

# Hugging Face API URL and API Key

API\_URL = "https://api-inference.huggingface.co/models/CompVis/stable-diffusion-v1-4" # Replace with your selected model API URL

HUGGINGFACE\_API\_KEY = os.getenv("hf\_FvMaFViSHdHxogILeeHGIkmyaTXGvOLVXN") # Store in environment variables

api = hf\_api.HfApi()

api.set\_access\_token(token)

if not HUGGINGFACE\_API\_KEY:

raise-ValueError("Hugging-Face-API-Key-not-found.-Set-the HUGGINGFACE\_API\_KEY environment variable.")

headers = {

"Authorization": f"Bearer {HUGGINGFACE\_API\_KEY}"

}

try:

response = requests.post(

API\_URL,

headers=headers,

data = {

"inputs": "A description of what you want to generate an image of",

"parameters": {

"num\_images": 2

}

}

)

response.raise\_for\_status()

print("Response JSON:", response.json())

except requests.exceptions.HTTPError as e:

print("HTTPError:", e.response.status\_code, e.response.text)

except Exception as e:

print("Error:", str(e))

# AWS S3 Configuration

AWS\_ACCESS\_KEY = os.getenv("AKIAT4GVRKB4FAU46RMN") # Store in environment variables

AWS\_SECRET\_KEY = os.getenv("vw8SIJ5fNzfaXohRFAp+7qyrH1eIe2afYR10Vt0M") # Store in environment variables

S3\_BUCKET\_NAME = "myawsbuc-1223" # Replace with your bucket name

if not AWS\_ACCESS\_KEY or not AWS\_SECRET\_KEY:

raise ValueError("AWS credentials not found. Set the AWS\_ACCESS\_KEY\_ID and AWS\_SECRET\_ACCESS\_KEY environment variables.")

# Initialize S3 client

s3\_client = boto3.client(

"s3",

aws\_access\_key\_id=AWS\_ACCESS\_KEY,

aws\_secret\_access\_key=AWS\_SECRET\_KEY

)

# Example token error check

try:

# Example API call to Hugging Face or S3 interaction for debugging

print("Testing S3 connection...")

response = s3\_client.list\_buckets()

print("S3 Buckets:", response["Buckets"])

except Exception as e:

print("Error with token or credentials:", str(e))

# Initialize the Translator

translator = Translator()

@app.route('/generate-image', methods=['GET', 'POST'])

def home():

img\_url = None

if request.method == 'POST':

prompt = request.form['text']

language = request.form['language']

# Translate the prompt to English if necessary

try:

if language != 'english':

translated = translator.translate(prompt, src=language, dest='en')

prompt = translated.text

except Exception as e:

return f"Error in translation: {e}"

# Prepare the payload

payload = {"inputs": prompt}

# Send the API request

try:

response = requests.post(API\_URL, headers=headers, json=payload)

if response.status\_code == 200:

# Save the image locally

image\_filename = "generated\_image.png"

with open(image\_filename, "wb") as f:

f.write(response.content)

# Upload the image to S3

try:

s3\_key = f"generated\_images/{image\_filename}"

s3\_client.upload\_file(image\_filename, S3\_BUCKET\_NAME, s3\_key)

img\_url = f"https://{S3\_BUCKET\_NAME}.s3.amazonaws.com/{s3\_key}"

except Exception as e:

return f"Failed to upload to S3: {e}"

return render\_template\_string(f"""

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Text to Image Generator</title>

</head>

<body>

<h2>Image Generated Successfully!</h2>

<img src="{img\_url}" alt="Generated Image" style="max-width:100%;"><br>

<p>Image saved to S3. <a href="{img\_url}" target="\_blank">View in S3</a></p>

<a href="/">Generate Another Image</a>

</body>

</html>

""")

else:

return f"Error: {response.status\_code}, {response.text}"

except requests.exceptions.RequestException as e:

return f"Request failed: {e}"

# Render the HTML template with the form

return render\_template\_string("""

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Text to Image Generator</title>

<style>

/\* Background image and page styling \*/

body {

font-family: baguet script, sans-serif;

background: url('https://st5.depositphotos.com/2274151/65350/i/450/depositphotos\_653505068-stock-photo-abstract-pastel-colors-smoke-background.jpg') no-repeat center center fixed;

background-size: cover;

height: 100vh;

display: flex;

flex-direction: column;

justify-content: center;

align-items: center;

text-align: center;

margin: 0;

}

/\* Header style \*/

h1 {

color: black;

font-size: 3em;

margin-top: 20px;

}

/\* Form container \*/

.form-container {

background-color: golden; /\* semi-transparent background \*/

padding: 20px;

border-radius: 8px;

box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);

width: 400px;

text-align: center;

}

/\* Input fields and select dropdown \*/

input[type="text"], select {

padding: 10px;

width: 100%;

margin-bottom: 20px;

border-radius: 4px;

border: 1px solid #ddd;

font-size: 1em;

text-align: center; /\* Center the text in the input field \*/

}

input[type="text"] {

background-color: white; /\* Remove background color \*/

}

/\* Submit button \*/

button {

padding: 12px 30px;

background-color: white;

color: black;

border: none;

border-radius: 4px;

cursor: pointer;

font-size: 1.2em;

}

button:hover {

background-color: white;

}

/\* Generated image section \*/

.generated-image {

margin-top: 20px;

}

.generated-image img {

max-width: 100%;

border-radius: 8px;

margin-top: 20px;

}

/\* Responsive design for mobile \*/

@media (max-width: 768px) {

.form-container {

width: 90%;

}

h1 {

font-size: 2em;

}

}

</style>

</head>

<body>

<h1>Text to Image Generator</h1>

<div class="form-container">

<form action="/" method="POST">

<label for="text">Enter Prompt:</label><br>

<input type="text" id="text" name="text" required><br><br>

<label for="language">Select Language:</label><br>

<select id="language" name="language" required>

<option value="english">English</option>

<option value="kn">Kannada</option>

<option value="ta">Tamil</option>

<option value="te">Telugu</option>

</select><br><br>

<button type="submit">Generate Image</button>

</form>

</div>

{% if img\_url %}

<div class="generated-image">

<h2>Generated Image:</h2>

<img src="{{ img\_url }}" alt="Generated Image">

</div>

{% endif %}

</body>

</html>

""")

if \_\_name\_\_ == '\_\_main\_\_':

app.run(app.run(host='0.0.0.0', port=8000, debug=True)

**6.Software Testing**

Software testing ensures that the Multi-Language Text-to-Image Generator functions as intended, meets the defined requirements, and performs efficiently under various conditions. The testing process involves validating both functional and non-functional aspects of the system.

* Ensure that the system generates accurate images from textual descriptions.
* Validate multi-language support, ensuring compatibility and correctness across supported languages.
* Confirm the smooth integration of components like AWS S3, Flask, and the machine learning model.
* Test the system’s usability, performance, and security.

**Integration Testing**

**Integration Testing** focuses on verifying the interaction between different modules or components of the system to ensure they work together seamlessly. For the **Multi-Language Text-to-Image Generator**, integration testing ensures that all interconnected components function as expected, including the frontend, backend, machine learning model, and AWS services.

**Objectives of Integration Testing:**

* Verify the correct interaction between user input and backend processing.
* Ensure proper communication between the Flask backend and the text-to-image model.
* Test the integration of AWS services, such as storing and retrieving images from S3.
* Detect and resolve interface-level issues between modules.

**Integration Testing Approach**

The **Hybrid Integration Testing Approach** is used, combining top-down and bottom-up methods:

1. **Top-Down Testing:** Tests high-level modules (e.g., user input and API communication) before moving to lower-level ones.
2. **Bottom-Up Testing:** Tests lower-level modules (e.g., ML model output) before integrating them with higher-level modules.

**Steps in Integration Testing**

**Identify Integration Points:**

* User input (frontend) and Flask API.
* Flask backend and the ML model.
* Backend and AWS S3 for image storage/retrieval.

**Design Test Cases:**

Test cases are written to validate the data flow and interaction between modules.

**Execute Test Cases:**

* Perform tests incrementally, starting with two integrated modules and gradually adding others.

**Validate Results:**

* Check if the output meets expected outcomes.

**Integration Scenarios:**

1. **Frontend and Backend:**
   * Scenario: User submits a text prompt and language selection.
   * Test: Verify that the data is received by the backend API and logged correctly.
2. **Backend and ML Model:**
   * Scenario: Backend sends pre-processed text to the ML model.
   * Test: Ensure the ML model returns a valid image or an appropriate error.
3. **Backend and AWS S3:**
   * Scenario: Save and retrieve generated images to/from S3.
   * Test: Verify that images are stored in the correct S3 bucket and can be retrieved.
4. **End-to-End Integration:**
   * Scenario: User enters text, selects a language, and generates an image.
   * Test: Ensure the image is displayed on the user interface, and the workflow completes without errors.

**Tools for Integration Testing**

* **Postman:** For testing API endpoints and data exchange.
* **Py-Test:** For writing and automating backend integration tests.
* **AWS SDK for Python (Boto3):** For testing AWS S3 integration.
* **Selenium:** For end-to-end UI integration testing.

**Challenges in Integration Testing**

* Synchronizing modules with asynchronous workflows, such as AWS S3 operations.
* Handling failures in external dependencies, like the ML model or translation APIs.
* Debugging complex interactions between modules.

A screenshot of a computer program

Description automatically generated

**7.Conclusion and Scope for Future Enhancement**

**Conclusion**

The **Multi-Language Text-to-Image Generator** is a robust and scalable system that leverages cutting-edge technologies to generate images from textual descriptions in multiple languages. The project integrates natural language processing (NLP), deep learning, and cloud computing to provide an interactive and user-friendly platform.

Key achievements of the project include:

* Seamless multi-language support for text inputs.
* Integration with AWS services like S3 for secure storage and EC2 for deployment.
* Scalable and modular architecture that ensures high performance and reliability.

This project demonstrates how advanced AI and cloud technologies can combine to create meaningful applications for diverse users. The system not only meets its functional requirements but also sets a foundation for innovation in AI-driven content creation tools.

**Scope for Future Enhancement**

While the current system is effective and functional, there are several opportunities for enhancement to make the solution more versatile and user-centric.

1. **Improved Model Capabilities:**
   * Use state-of-the-art models for better image quality and creativity.
   * Incorporate fine-tuning mechanisms to adapt the system for specific domains or use cases, such as fashion, education, or gaming.
2. **Enhanced Multi-Language Support:**
   * Expand language coverage to include more regional and indigenous languages.
   * Improve translation accuracy by integrating advanced language models.
3. **User Personalization:**
   * Allow users to customize output images with additional parameters like style, color, and layout.
   * Implement user profiles to save preferences and previous projects.
4. **Mobile Application Integration:**
   * Develop a mobile application version for easier accessibility and usability.
5. **Integration with Other Platforms:**
   * Enable sharing of generated images directly to social media platforms.
   * Provide APIs for third-party applications to integrate text-to-image capabilities.
6. **AI-Powered Insights:**
   * Incorporate AI-based feedback on input text for better image generation.
   * Provide suggestions for alternate descriptions or refinements.
7. **Performance Optimization:**
   * Optimize image generation times, especially for high-resolution outputs.
   * Use serverless computing with AWS Lambda for cost-effective scaling.
8. **Security Enhancements:**
   * Implement advanced authentication mechanisms to ensure data security.
   * Encrypt images and metadata stored in AWS S3 for enhanced privacy.
9. **Real-Time Collaboration:**
   * Introduce features for collaborative projects where multiple users can contribute text descriptions or make edits to the generated images.
10. **Research and Development:**
    * Explore integration with AR/VR technologies to create immersive experiences.
    * Study user feedback and evolving AI advancements to continuously enhance the system.

**Bibliography**

**Online Documentation and Resources**

1. **Flask Official Documentation:**
   * https://flask.palletsprojects.com/
   * Provided insights into building and deploying a Flask application.
2. **AWS S3 Documentation:**
   * <https://docs.aws.amazon.com/s3/>
   * Detailed information on securely storing and retrieving images.
3. **AWS EC2 Documentation:**
   * <https://docs.aws.amazon.com/ec2/>
   * Guide for deploying the application on EC2 instances.
4. **Python Official Documentation:**
   * <https://docs.python.org/>
   * Reference for Python libraries and best practices.
5. **Hugging face API Model Documentation:**

* For understanding the capabilities and constraints of text-to-image generation models.

**7.SNAPSHOTS**

A screenshot of a computer

Description automatically generated

**Command:**  
pip install -r requirements.txt

**Description:**  
This command installs all Python packages listed in a file named requirements.txt. It is commonly used to set up project dependencies in a single step, ensuring all necessary packages and their versions are installed.

A screen shot of a computer

Description automatically generated

With the command-python app.py(app.py is the flask app code with python language)

I got <http://127.0.0.1:5000> url

This URL http://127.0.0.1:5000 typically represents the local address of a web application running on your computer. Here's a proper description:

**Description:** This URL is the default local address (127.0.0.1, also known as localhost) used to access a web application hosted on your computer. The :5000 at the end specifies the port number the application is using. It is commonly associated with Flask applications or other development servers. Accessing this address in a browser will allow you to interact with the application running on your local machine.

A screenshot of a computer

Description automatically generated

when the url is pasted in the web browser this web page will be displayed where I can input text as a prompt and image is generated.

A painting of a sun with a face

Description automatically generated

This is the generated image, the image is stored in s3 bucket and there is “Generate another image” option where the backend code takes you to back to the frontend page to generate another image

* **Now steps to deploy into ec2:**

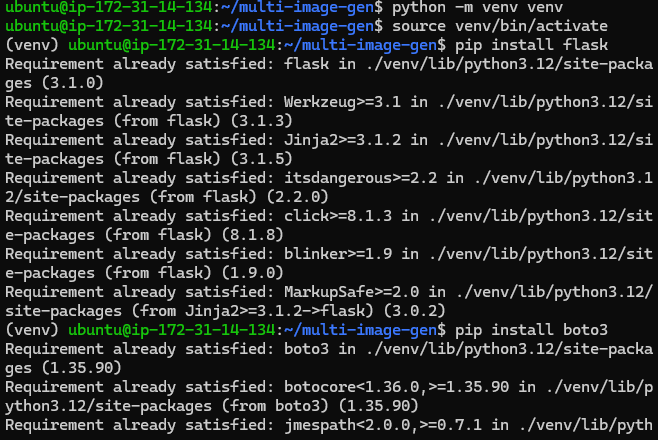
A computer screen shot of a black screen

Description automatically generated

**ssh -I “path/to/your/key” ubuntu@<ip\_address>**

This command is used to securely connect to a remote server via SSH (Secure Shell) using a private key for authentication. Here's a breakdown of each part:

* **ssh:** The command to initiate an SSH connection.
* **-i "path/to/your/key":** Specifies the path to the private key file (usually .pem format) required for authentication.
* **ubuntu@<ip address>:** The username (ubuntu in this case) and the IP address of the remote server you want to connect to.



**Command:**  
python -m venv venv

**Description:**  
This command creates a new virtual environment for Python projects, allowing you to manage dependencies separately from the global Python environment. Here's a detailed breakdown:

* **python:** Refers to the Python interpreter. It could be python3 if you're explicitly using Python 3.
* **-m venv:** This uses Python's venv module to create a virtual environment.
* **venv:** This is the name of the folder where the virtual environment will be created. You can replace venv with any folder name you prefer.

**Command:**  
source venv/bin/activate

**Description:**  
This command is used to activate a Python virtual environment in a Unix-based system (e.g., Linux or macOS). Once activated, the virtual environment isolates your Python project’s dependencies from the global Python environment.

**Breakdown:**

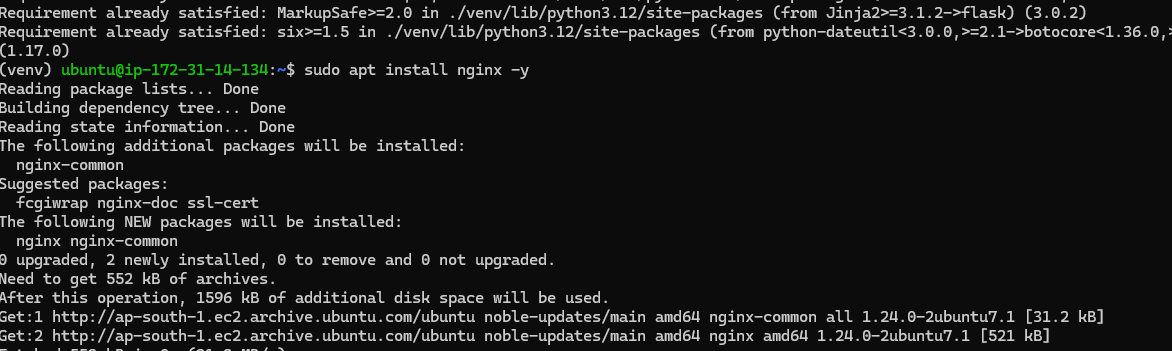
* **source:** A shell command to execute the script in the current shell session.
* **venv/bin/activate:** The path to the activation script for the virtual environment. This script adjusts your shell environment so that Python commands and package installations are directed to the virtual environment.

**Command:**  
pip install boto3

**Description:**  
This command installs the boto3 library, which is the official Amazon Web Services (AWS) Software Development Kit (SDK) for Python. boto3 provides a simple, Pythonic way to interact with AWS services such as S3, EC2, DynamoDB, Lambda, and more.

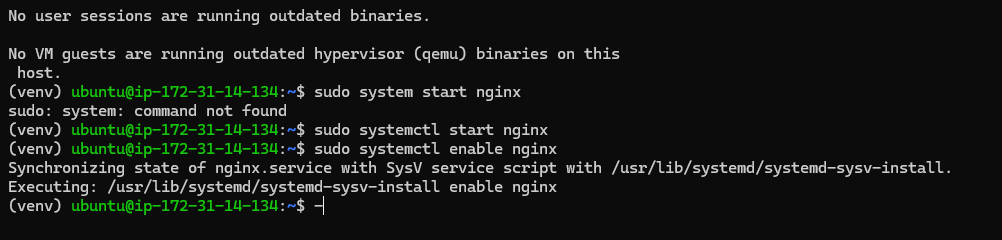
**Breakdown:**

* **pip:** The Python package installer.
* **install:** The action to install the specified library.
* **boto3:** The name of the library to be installed.



**Command:**  
sudo apt install nginx -y

**Description:**  
This command installs **Nginx**, a high-performance web server and reverse proxy server, on a Debian-based Linux system (e.g., Ubuntu). The -y option automatically confirms the installation prompt, saving time during the installation process.



**Command:**  
sudo systemctl start nginx

**Description:**  
This command starts the **Nginx** web server on your system using the systemctl utility. It is useful for manually starting Nginx after installation or if it is stopped for any reason.

**Command:**  
sudo systemctl enable nginx

**Description:**  
This command configures **Nginx** to start automatically at system boot using the systemctl utility. It ensures that the Nginx service runs whenever the system is restarted, making it ideal for production environments.

A computer screen shot of a program

Description automatically generated

**Command:**  
pip install googletran

**Description:**  
This command installs the googletran library, a Python package for interacting with Google Translate's unofficial API to translate text between languages. However, note that googletran may be outdated or unreliable due to changes in Google Translate's API. The recommended alternative is googletrans==4.0.0-rc1, which is more actively maintained.

A computer screen with white text

Description automatically generated

**File**   
/etc/nginx/conf.d/flask\_app.conf

**Description:**  
This is a typical location for a **Nginx configuration file** that is used to set up a Flask web application with Nginx as a reverse proxy server. The flask\_app.conf file contains the specific configuration directives that tell Nginx how to handle HTTP requests and forward them to the Flask application, which is usually run by a WSGI server like **Gunicorn**.

A screenshot of a computer program

Description automatically generated

We can also write code from ubuntu directory by opening app.py file from command

**Nano app.py.**

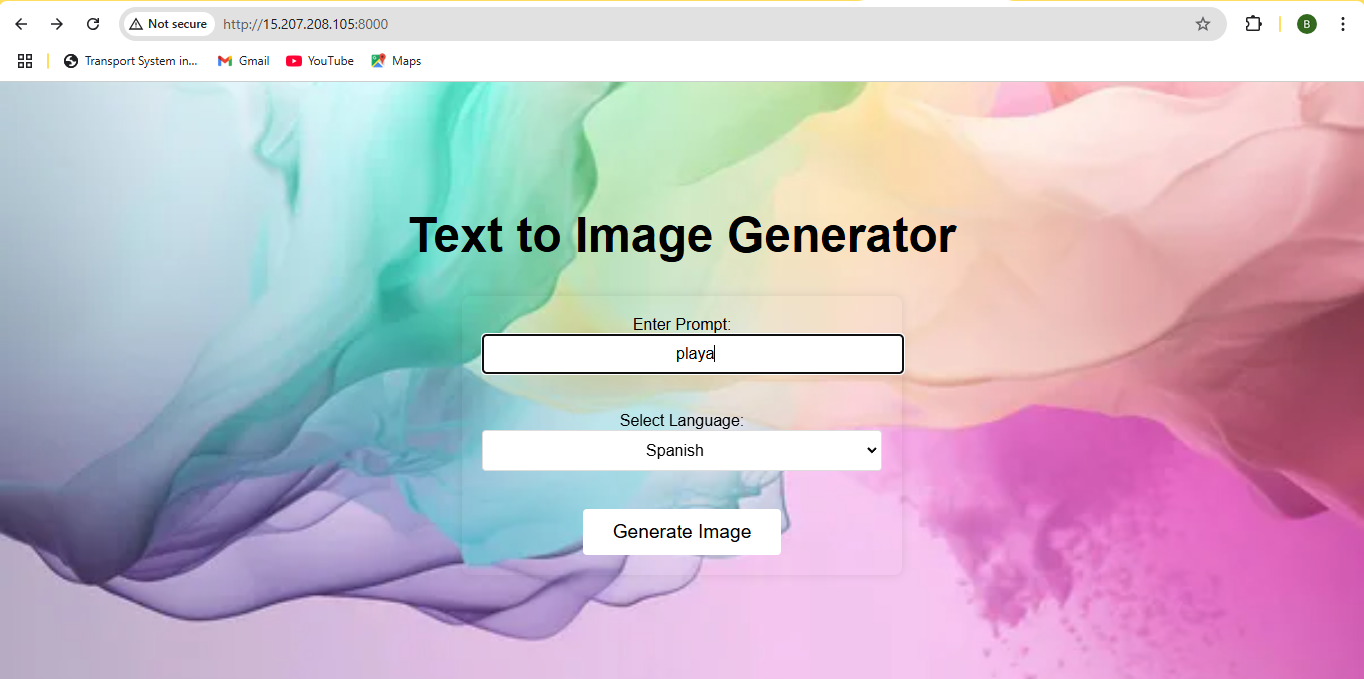
A screenshot of a computer program

Description automatically generated

**Command:**  
gunicorn -w 4 -b 0.0.0.0:8000 app:app &

**Description:**  
This command runs a **Flask** application using **Gunicorn**, a WSGI HTTP server for Python web applications. It specifies multiple options to configure the number of workers, the binding address and port, and the target Flask app.

We can close this command by **ctrl+c**



After the gunicorn command is given the app starts to run successfully, we can go to new web browser to open our web app by giving **- http://<ec2\_public\_ip>:8000**

Once this is given, we will get our web app page.

A screenshot of a computer screen

Description automatically generated

This is the final output after accessing the web app from ec2 public address and entering the prompt.

Above is the image of **playa** in **Spanish,** which is beach**.**