IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 3, November 2025
Impact Factor: 7.67

Text-to-Image Generator Using MERN Stack

Prof. Bhramdeo Wadibhasme, Ms. Divya Zade, Ms. Shejal Burele
Ms. Bhagyashree Chepurwar, Ms. Vanshika Akanpalliwar
Department of Computer Science & Engineering
Tulsiramji Gaikwad Patil College of Engineering and Technology, Nagpur, Maharashtra, India.
bramhadeo.ece@tgpcet.com, divyazade82@gmail.com
shejalburele2005@gmail.com, bchepurwar091@gmail.com, vanshikaakanpalliwar@gmail.com

Abstract: *Imagify is a text-to-image generator that uses the MERN stack -MongoDB, Express.js, React.js, and Node.js- to create a user-friendly, scalable, and real time environment for turning text descriptions into images. It combines a transformer- based model for understanding language with either a diffusion or GAN- based network for generating images, allowing it to create visual that accurately match the meaning of the input text with high quality.*

In Imagify, the frontend is built with React.js, which makes the user experience smooth, responsive, and allows for real- time previews as users input their prompts. The backend runs on Node.js and Express.js, which manage the API calls, run the image generation processes, and keep the frontend connected to the AI model. MongoDB is used to store user input, image links, and usage data which helps keep the system organize and scalable. This setup enables multiple users to access and use the system at the same time with very little delay. Tests show that imagify can consistently produce images that are both meaningful and visually correct, even with a wide range of prompts.

It has strong potential for use in areas like digital art, teaching tools, content creation, and helping people who may not have artistic skills. Overall, Imagify is a great example of combining modern web development with deep learning models, offering a dependable platform for generating images from text and showing the future of AI- powered creativity.

Keywords*:* MERN Stack; MongoDB; Express.js; React.js; Node.js; Full-Stack Web Development; Web-Based Application; API Integration; Cloud Database; User Interface Design

I. INTRODUCTION

In recent years, there has been a big increase in the need for interactive and automated digital content creation, especially in areas like media design, education, advertising and entertainment Text-to image-generation tools helps with this by letting users create pictures by simply describing what they want in everyday language. But a lot of the existing platforms are hard to use, complicated to access, or don't have an easy way for users to interact with the image-making system in real time.

Imagify is a web-based tool that turns text into images, built using the MERN stack-meaning it uses MongoDB, Express.js, React.js, and Node.js. The MERN stack was chosen because it's flexible, can handle growth, and help the website and server talk to each other smoothly. In Imagify, React.js makes the user interface fast and easy to use, so user can type in their ideas and see the images right away. Node.js and Express.js take care of the back-end work, handle request from the website, and make sure information is shared securely and quickly. MongoDB keeps track of what users ask for, the images they create, and the system's activity, which helps in organizing and finding data easily.

By putting text-to-image generation on a MERN-based website, imagify makes it easier for regular people, designers, and students to use AI tools for creativity.

The goal of imagify is to make it simple for users to work with complex image-making systems, letting them create images in real time without needing special software or technical knowledge. This project shows that web development and AI can work well together to make a useful and interactive creative tool.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 3, November 2025 Impact Factor: 7.67

The fast development of artificial intelligence (AI) has allowed machine to understand and copy creative tasks that humans usually do. One big breakthrough is text-to-image generation, where a written description is turn into a picture. This technology has grown a lot because of better generative modelling, which lets people create high-quality images from their descriptions. These systems are useful in many areas like digital art, education, entertainment, virtual environment, marketing, and helping people who don't have strong design skills.

Even though research on Text- to-image generation has expanded, these systems are still not very easy to use. Many platforms need hardware, complicated setups, or technical know-how. Users also struggle with handling inputs, saving images, or working with models in real time. For these tools to be really useful, they should be part of platforms that simple, fun to use, and available to everyone.

To create a web- based platform where users can generate images from descriptions written in a natural language, in a simple and easy-to-use way.

To build a full- stack application using the MERN stack (MongoDB, Express.js, React.js, Node.js) that is scalable, efficient, and easy for users to navigate.

To design a user-friendly interface with React.js that lets users enter their image ideas, see the generated images, and keep track pf their past creations.

To set up a secure and fast backend with Node.js and Express.js that handles image generation requests processes text inputs, and sends the resulting images quickly.

II. LITERATURE SURVEY

Text-to-image synthesis has developed quickly in the past ten years, shifting from using condition GANs to bigger autoregressive and diffusion- based models that offers much better image quality and alignment between text and images. Early methods used conditional and stacked GANs to create images based on captions; StackGAN and AttnGAN added multi-stage generation and attention mechanisms to improve how well words match parts of the image showing attention and multiple stages help create accurate images and finer details.

A different line of research looked at autoregressive modelling of image tokens with text. These autoregressive transformer methods treated text and images tokens together and achieved strong results in generating images without seeing examples, proving that scaling transformers to handle both text and image tokens can create varied output without needing special architecture.

Another important approach combined strong multimodal encoders like CLIP with generative decoders or optimization methods. Methods like VQGAN+CLIP use a discrete image generator (VQGAN) that is guided by CLIP's joint text and image embeddings to make high- quality images from text prompts- this flexible approach led to lots of experimentation and practical tools for image generation.

Diffusion models have been the biggest recent breakthrough in text- based image creation. Denoising Diffusion Probabilistic Models (DDPM) introduced a scalable and high- quality way to generate images, and later, they were used for text-conditioned images synthesis. Latent diffusion techniques, like the Stable Diffusion family moved the diffusion process into a lower- dimensional latent space to cut down on computing power and memory needs without losing image quality, making practical open- source text-to-image models possible and widely used. Independent research, like Imagen, showed that combining large language encoders with diffusion models leads to very good alignment between images and text, and realistic images on standard tests.

Recent improvement, such as SDXL and imagen 3, focus on better conditioning methods, multi-stage decoders, and safer practices. These show that model size, careful conditioning (like using bigger or dual text encoders), and varied training data greatly affect both image quality and alignment. Surveys of text-to-image diffusion models cover these trends and common ways to evaluate them.

Researchers often test text-to-image systems on standard datasets like COCO using numerical scores (FID, CLIP-score) and human evaluations (pairwise tests, DrawBench). Human feedback is still important because automated scored don't always capture how well an image matches the text in meaning or structure.

Even though the best generative models make very high- quality images, many require a lot of resources or are only available through hosted APIs. The rise of latent diffusion an open- source tools like stable diffusion had made it easier

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 3, November 2025 Impact Factor: 7.67

to use these models, allowing them to be integrated into web application. Real- world web system must handle model interface (local vs cloud GPU), speeds, API management, storing prompts and results, and user experience. Putting such complex models into a full web app bring up engineering challenges regarding scalability, caching, and data handling- exactly the gap the solution like imagify aim to fill by wrapping model use in secure Node/Express endpoints and letting users interact with the app through React, while using MongoDB to store metadata and history.

III. METHODOLOGY OF THE SYSTEM

The proposed project, Text-to-Image Generator using the MERN Stack, aim to create realistic images from natural language text by combining deep learning models for image creation with a web-based user interface built using the MERN (MongoDB, Express.js, React.js, Node.js) technology stack. This section explains the system's architecture, workflow, functional modules, integration process, and evaluation methods that define how the system works and is implemented. The project uses a modular approach where the frontend collect the user's text input, the back-end processes the request through APIs connected to a text-to-image model (such as Stable Diffusion, Clipdrop, or DALL.E API), and the generated image is sent back, stored, and displayed on the web interface.

System Architecture: The overall structure of the Tex-to-Image Generator system is divided into five main layers:

1. User interface layer (Front-End) – React.js

This layer handles all user interactions: It lets users enter text prompts, shows the generated images and related data likes the prompt, timestamp, and image resolution. It also uses a responsive design to work well on both desktop and mobile devices.

2. Application Layer – Node.js and Express.js (Back-End)

This layer acts as a middleman between the front-end and the AI model. It routes the request, manages API keys securely, and handles user authentication. It also deals with making asynchronous API calls to the external AI model or local interface engine. It provides RESTful API is to send and receive data between the client and server.

3. Model Layer – Text-to-Image Generation Engine

This is the core part of the system that creates images based on the text input. It can use either a hosted API (like Clipdrop, OpenAI, DALL.E, or Stability.ai) or a locally delayed model (like Stable Diffusion). It uses pretrained deep neural networks such as CLIP for text-image embedding and Latent Diffusion Models (LDM) for image synthesis.

4. Database layer – MongoDB

This layer stores user data, text prompts, image metadata, and usage logs. It keeps references to image URLs stored (like AWS S3 or Firebase). It helps quickly retrieve images for display and supports statistical analysis for research evaluation.

5. Cloud or Server Layer

This layer hosts the MERN application and model service on a cloud platform (like AWS, Render, or Vercel). It ensures scalability, security, and real-time access. It also uses load balancing to handel multiple simultaneous requests.

My name is shejal vijay burele. The generated image saves the image metadata, including the prompt generation time, model parameters, and user ID, into MongoDB. The image file is stored in a cloud bucket or locally, with reference saved in the database.

The React front-end fetches the image URL and display it in the user interface. Users can download, rate, or regenerate image using update prompts. They can also provide feedback for evaluating the model's

The front-end (React.js) sends the input to the Node.js/Express.js server via an API ca

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 3, November 2025
Impact Factor: 7.67

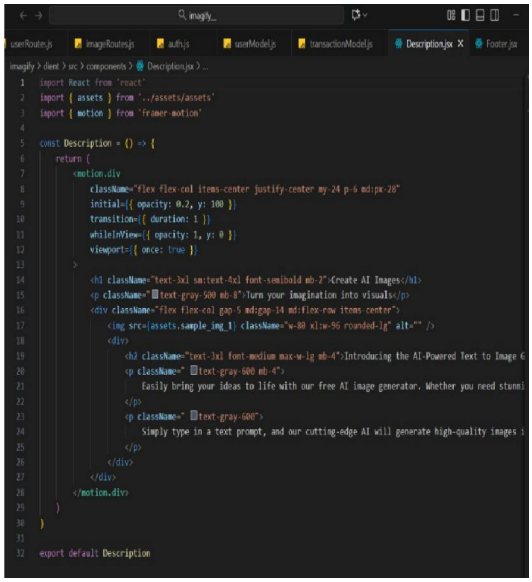

Fig-1: Code

A. Workflow of the System: The text-to-image generation process follow these steps.

1. User Input

The user enters a description of the desired image like "a mountain landscape at sunset with a lake in front." They can also provide optional derails like image size, style, or color.

2. Preprocessing and Request Handling

The front-end (React.js) sends the input to the Node.js/Express.js server via an API call. The back-end checks and cleans the prompt to prevent security issue or inappropriate content.

3. Model Invocation

The server sends the processed text prompt the text to-image model. If using a hosted API, it makes an HTTP POST request to the model endpoint (such as Stability or Clipdrop). If using a local model the text is encoded using a CLIP text encoder, and the diffusion model creates the image in latent space.

4. Image Generation Process

The text prompt is converted into an embedding vector that represent its meaning. The model then refines random noise in multiple steps to create an image that matches the prompt's meaning. Parameters like guidance scale and random seed control image variety and accuracy.

5. Result Handling and Storage

The generated image saves the image metadata, including the prompt generation time, model parameters, and user ID, into MongoDB. The image file is stored in a cloud bucket or locally, with reference saved in the database.

6. Display and Feedback

The React front-end fetches the image URL and display it in the user interface. Users can download, rate, or regenerate image using update prompts. They can also provide feedback for evaluating the model's performance.

B. Flowchart

The text prompt is converted into an embedding vector that represent its meaning. The model then refines random noise in multiple steps to create an image that matches the prompt's meaning. Parameters like guidance scale and random seed control image variety and accuracy.

The text prompt is converted into an embedding vector that represent its meaning. The model then refines random noise in multiple steps to create an image that matches the prompt's meaning. Parameters like guidance scale and random seed control image variety and accuracy.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 3, November 2025
Impact Factor: 7.67

The text prompt is converted into an embedding vector that represent its meaning. The model then refines random noise in multiple steps to create an image that matches the prompt's meaning. Parameters like guidance scale and random seed control image variety and accuracy.

The text prompt is converted into an embedding

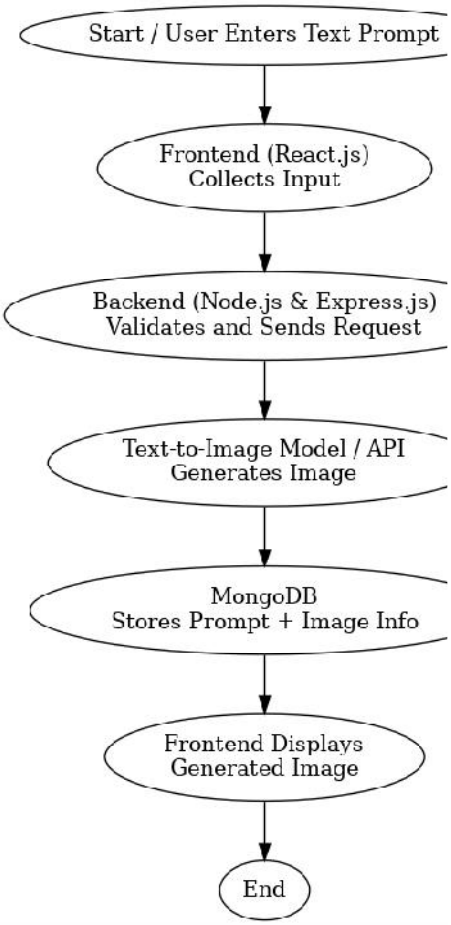

Fig-2: Flowchart

C. Functional Modules of the System: The system has the following key modules.

1. Text Input Module

This module accepts user provided text descriptions. It offers input validation and basic prompt suggestions.

2. Image Generation Module

This module converts the validated text input into an image using the deep learning model or API call. It uses techniques like prompt embeddings, attention mechanisms, and latent diffusion for accurate image generation.

3. Database Management Module

This module manages the connection between prompts and images in MongoDB. It implements create, read, update, and delete operations and uses indexing to allow quick data retrieval.

4. User Interface Module

This module display images in a user-friendly way. It allows searching, filtering, and reusing past prompts.

5. Feedback and Evaluation Module

This module collects user feedback or ratings to assess the quality of the generated images. It calculates metrics like FID (Frechet Inception Distance) and CLIPScore for performance analysis.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 3, November 2025 Impact Factor: 7.67

6. Security and Authentication Module

This module ensure secure handling of API key and user authentication via JWT tokens. It prevents misuse of the image generation endpoints.

IV. IMPLEMENTATION

The Imagify system is built using the MERN stack, with combines MongoDB, Express.js, React.js, and Node.js to create a single full-stack environment. It uses a modular and scalable setup, making it easier to develop and maintain different parts like the user interface, server logic, and data handling. The system is structured into four main parts the frontend interface, backend services, model interface layer, and database storage.

1. Frontend Implementation (React.js)

The frontend is created with React.js, which allows for a smooth and interactive user experience. There's a specific area where users can type in a description for the image they want to generate. This form connects to React hooks that keep track of user input and send request to the backend. Once the user clicks submit, the fronted shows them real-time updates like loading signs or error messages if something goes wrong. The image that is created is shown right there, and there's also a history section that shows past images from the backend. React component system helps keep the prompt from, image display, and history panel as separate, reusable parts of the UI.

2. Backend Implementation (Node.js and Express.js)

The backend is built with Node.js, using Express.js for handling the routing. It provides RESTful API endpoints for sending prompts and getting the history of generated images. The system checks that the prompt is not empty and doesn't contain harmful content. If it passes, the prompt get sent to the model inference layer. The backend also manages things like timeouts and request limits to make sure the system stays stable and predictable. Express middleware is used to handle security stuff like cross-origin resource sharing, which makes the system safer and more compatible with other services.

3. Model Inference Layer

The model inference layer is where the text prompt becomes an image. This part can use either a local model like Stable Diffusion or a cloud-based service like Stability AI, Replicate, or OpenAI's image API. In both cases, the backend sends the prompt and any needed settings to the model. The model then turns the text into an image through natural language processing and image creation techniques. The image can come back as a URL, aa baase64 string, or a local file path. The backend standardizes this information before sending it back to the frontend and saving it to the database.

4. Database Layer (MongoDB)

All the generated images and related prompts are stored in MongoDB. Each entry has the text the user typed, the generated image (either a URL or base64), details about the model used, and a timestamp using MongoDB makes it easy to add new fields in the future without changing existing data, which helps when testing new models, keeping track of user feedback or analyzing how prompts are reused.

5. System Workflow Summary

A user types in a description in the React interface. This prompt is sent to the backend via a POST request. The backend checks the input and sends it to the model inference layer. The model creates the image and send it back to the backend. The backend then saves both the image and the prompt in MongoDB. The frontend then gets the image from the backend and display it along with updating the history panel.

V. RESULTS AND ANALYSIS

After successfully creating the text-to-image generator using the MERN stack, the system was checked based on how easy it is to use, and the quality of the images it produces main goal of this check was to see how well the system turns into clear, meaningful images and how well software structure supports this process.

Functional Output Evaluation

The system was tested with many different types of text prompts. These included simple descriptions of objects, real-life scenes, artistic drawings, and abstract ideas. For clear and specific prompts, the system made images that closely

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 3, November 2025 Impact Factor: 7.67

matched what was expected. For example, when someone asked for "A white cat sitting on a window sill," the image clearly showed the cat, looked natural, and fit well into the scene.

When the prompts were more abstract or symbolic, like "Freedom as a bird breaking through clouds," the system made creative and expressive images that matched the emotion and meaning of the input. This shows the system can handle both straightforward and imaginative ideas. It was noticed that the more detailed the prompt, the better the image result. If the prompt was too vague or short, the image tended to be more general.

System Performance Evaluation

The system's performance was checked by looking at how long it took to create images, how smoothly users could interact with it, and how stable the system was when handling multiple requests. On average, images were created in just a few seconds, depending on how complex the prompt was and how busy the model was. The front end stayed smooth and responsive because of the efficient state management in React and the asynchronous requests in Node.js.

Storing and retrieving images from MongoDB was fast and reliable, so user could easily see their past images without any lag. Even with many users using the system at the same time, the backend kept performing well, showing that the system is scalable and dependable.

User experience Assessment

To check how easy the system was to use, a group of users tried the app and gave their feedback. Most users found the interface clean, simple, and easy to use. They liked how straightforward it was to enter a prompt and get an image. The ability to review past images was also appreciated, as it made experimenting and reusing images easier.

Users generally found the image quality good, especially with detailed prompts. Some mentioned that very complex prompts took a bit longer to process, but they thought it was still acceptable. Overall, users were very satisfied with how the system worked, how quick it was, and how creative the images were.

Output Quality Interpretation

The images created by the system were checked based on how well they matched the prompt, how clear the details were, and how creative they were. Usually, the images showed what the prompt was asking for the detail clarity was better when the image had one object or a simple scene, while images with many elements or fine details sometimes had small flaws or blending issues.

For abstract or emotional prompts, the system showed a lot of creativity, making artistic images instead of literal ones. This suggests the model can do both realistic images and more expressive, stylistic artwork. However, how clear and accurate the image was strongly depended on how well the prompt describe what was needed.

The images created by the system were checked based on how well they matched the prompt, how clear the details were, and how creative they were. Usually, the images showed w hat the prompt was asking for the detail clarity was better when the image had one object or a simple scene, while images with many elements or fine details sometimes had small flaws or blending issues.

For abstract or emotional prompts, the system showed a lot of creativity, making artistic images instead of literal ones. This expressive, stylistic artwork. However, how clear and accurate the image was strongly depended on how well the prompt describe what was needed

The images created by the system were checked based on how well they matched the prompt, how clear the details were, and how creative they were. Usually, the images showed what the prompt was asking for the detail clarity was better when the image had one object or a simple scene, while images with many elements or fine details sometimes had small flaws or blending issues.

For abstract or emotional prompts, the system showed a lot of creativity, making artistic images instead of literal ones. This suggests the model can do both realistic images and more expressive, stythe imaig-

The images created by the system were checked based on how well they matched the prompt, how clear the details were, and how creative they were. Usually, the images showed what the prompt was asking for the de tail clarity was

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 3, November 2025
Impact Factor: 7.67

better when the image had one object or a simple scene, while images with many elements or fine details sometimes had small flaws or blending issues.

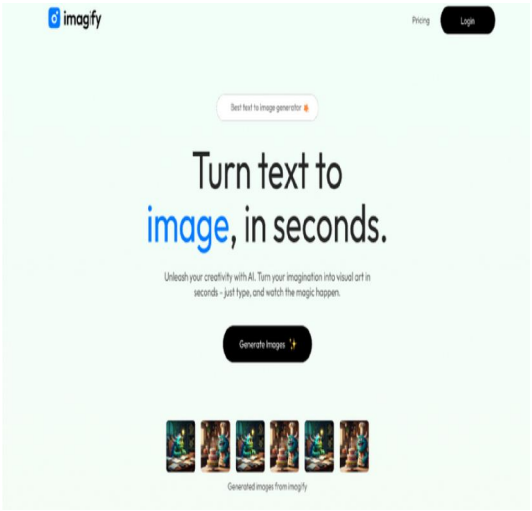

Fig-3:Interface

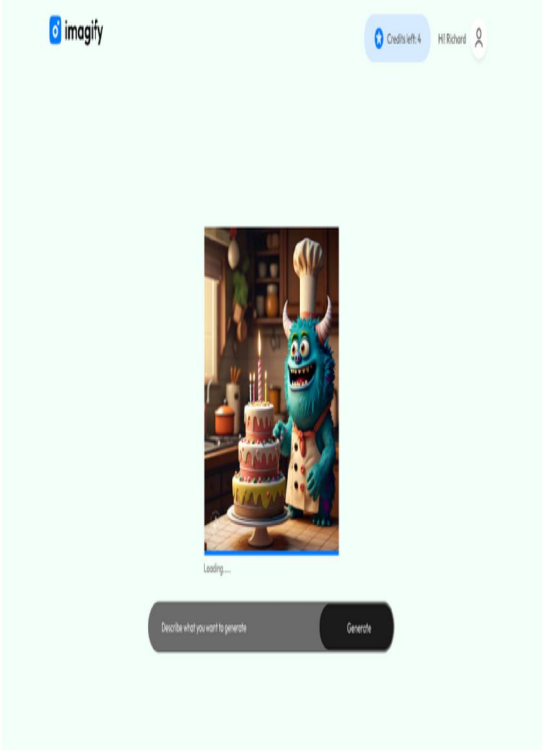

Fig-4: Image Generation Page

For abstract or emotional prompts, the system showed a lot of creativity, making artistic images instead of literal ones. This suggests the model can do both realistic images and more expressive, stylistic artwork. However, how clear and accurate the image was strongly depended on how

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 3, November 2025
Impact Factor: 7.67

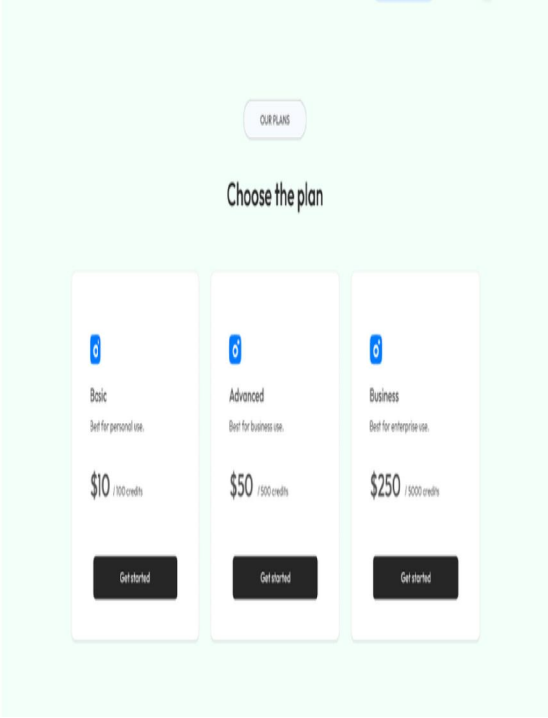

Fig-5: Credit Page

VI. FUTURE SCOPE

The Imagify system works well at turning text into images, but there are still several ways. It can get better to make it more useful, accurate, and enjoyable for users. As text-to-image technology keeps improving, the system can take advantage of these new developments to perform better and be used in more situations. One big area to focus on is using more than one image generation model. Right now, the system uses just one model.

If it could support different models, like realistic, anime, sketch, watercolour, 3D, or other, users would have more options to match their creative goals. This would let them pick the style that fits their idea best. Another thing the system could add is tools to edit and improve image after they're created. Users might want to change colours, tweak details, boost quality, or mix elements from different images. Features like editing specific parts of an image refining it based on new prompts, and using AI to make images sharper would give users more control over the final result. The system could also include user accounts and could storage.

This would let people save their image collections, keep track of the prompts they've used, and work together on projects. That would change Imagify from a simple tool into a full creative space. Another improvement would be training the model with data that's specific to different fields. For instance, industries like education, animation, game design, advertising, medical imaging, and storytelling could use tailored training to create images that are exactly what professionals need in those areas.

Lasting, the system could be made more scalable by using cloud-based solutions like Kubernetes or Docker. This would help handle many users at once and work smoothly in real-world settings where lots of people need to use it at the same time.

VII. CONCLUSIONS

The Imagify system shows how practical and efficient it can be to create images from text using the MERN stack-MongoDB, Express.js, React.js, and Node.js. By connecting a text-to-image API with a modern web framework, the system gives users an easy and interactive way to turn descriptive text into visual content. React.js makes the interface

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429 Volume 5, Issue 3, November 2025 Impact Factor: 7.67

fast and responsive, while Node.js and Express.js handle the backend, ensuring secure and scalable performance. MongoDB helps manage all the data, including the images and user input.

The results show that Imagify does what it was meant to do: let users create high-quality images based on their descriptions.

The project shows how combining Al image generation with full-stack web tech can lead to useful creative tools. Even though image quality can depend on how clear the prompt is and the model's limits, the current system is a good starting point for future upgrades and experimentation.

All in all, Imagify adds value to the field of Al-assisted creativity by offering a simple way to generate visual content. With more work on image quality, customization, model training, and scalability, the system could become more powerful and widely used in areas like digital art, education, media, and design.

The Imagify text-to-image generator is a working example of how to turn user text into visual outputs using the MERN stack. It shows how web development and Al image generation can work together.

React.js makes the interface smooth and easy to use, Express.js and Node.js handle the backend efficiently, and MongoDB keeps everything organized and secure. This setup lets users create, view, and save Al images without needing technical skills.

System testing shows that the app works well across different prompts, with image quality and relevance largely depending on how clear the text input is. Even though it uses external Al models, the design makes it easy to switch to more advanced or specialized models later. Performance tests also show that the MERN stack is flexible and can handle growth, making it good for both personal and team use.

The project also points out the growing need for Al-driven creativity tools in areas like digital art, marketing, e-learning, game design, and content creation. Imagify is an example of how these tools can be made available through web apps, helping more people use them. However, there's still room for improvement, such as better image quality, faster generation, more customization options, and real-time tweaks.

In conclusion, Imagify is a useful step forward in the field of Al-powered creative tools, combining solid web practices with new image generation tech.

With more development and model improvements, it has the potential to be a valuable tool in both creative and professional settings.

VIII. ACKNOWLEDGEMENT

The authors want to thank the open-source software communities for their important contributions. The tools, frameworks, and libraries created and kept up by these groups were the base of this research. Especially, the MERN stack technologies—MongoDB, Express.js, React.js, and Node.js—were very important in making the backend processing efficient, the interface interactive, and the data management secure. Because these tools are freely available, they made it easy to integrate and develop the web-based text-to-image generation platform quickly.

We also want to thank the developers and researchers behind public text-to-image generation APIs and model-based services. Their work on neural image creation, understanding user prompts, and visual creativity had a big impact on the design and how the project was built. Their progress in AI-powered content creation was the key to turning text into meaningful images.

We are also thankful to the academic and research groups working on human-computer interaction, creative computing, and digital art. Their published work on usability, design ideas, and interactive visualization helped shape the system's workflow, user experience, and how we evaluated the project.

We are grateful to the people who tested our system and gave feedback. Their helpful comments improved how the interface responded, how clear the images were, and how reliable the system was.

Finally, we thank the users who tried out the platform and gave a variety of prompts for image generation. Their involvement helped show how useful and flexible the Imagify platform is. The work of open-source developers, AI researchers, testers, and early users together made this project possible, turning it into a working system that serves users. Their efforts keep pushing the field of smart creative applications forward.

IJARSCT
International Journal of Advanced Research in Science, Communication and Technology
International Open-Access, Double-Blind, Peer-Reviewed, Refereed, Multidisciplinary Online Journal
ISSN: 2581-9429
Volume 5, Issue 3, November 2025
Impact Factor: 7.67

REFERENCES

1. ClipDrop API Documentation: ClipDrop API is an API for creating images and editing pictures, made by Stability AI, the same company that made Stable Diffusion. It lets developers make images, take out backgrounds, make images bigger, improve lighting, change the sky, and do lots of other creative things through easy API calls. This makes it really helpful for web apps like your Imagify MERN project.
URL: https://clipdrop.co/apis

2. React Official Documentation: React.js is an open-source JavaScript library created by Facebook to help build dynamic and interactive user interfaces, particularly for single-page applications. It uses a component-based structure, which breaks down the user interface into smaller, reusable parts called components. This approach makes the application easier to manage, reuse code, and scale as needed.
URL: https://react.dev

3. Express.js Documentation: Express.js is the backend framework that runs on Node.js. It connects the frontend, which is built with React.js, to other backend services like APIs from ClipDrop and databases like MongoDB.
URL: https://expressjs.com

4 . MongoDB Documentation: MongoDB is a type of NoSQL database that works with documents, and it's used in the MERN stack to save data in a format similar to JSON. In the Imagify project, MongoDB is mainly used to handle and keep data stored permanently.
URL:https://www.mongodb.com/docs/

5. Razorpay Developer Guide: Razorpay is a payment gateway based in India that enables businesses and apps to receive online payments. It offers support for several payment methods including credit and debit cards, UPI, net banking, and wallets. Additionally, it provides features for handling subscriptions and recurring payments, as well as tools like payment links and checkout forms.
URL: https://razorpay.com/docs

6 . Tailwind CSS Documentation: Tailwind CSS is a CSS framework that focuses on providing small, reusable classes to help developers create modern, flexible, and easily adjustable user interfaces. Rather than using ready-made components, it lets you build styles by combining these utility classes directly in your HTML.
URL: https://tailwindcss.com

7 . Vite Official Documentation: Vite is a new tool for building frontend websites that makes development quicker and easier, especially for projects using React, Vue, or other similar frameworks. Unlike older tools like Webpack, Vite uses the same built-in JavaScript features in the browser during development and has a faster way to prepare the final website for the web.
URL: https://vitejs.dev