1. INTRODUCTION

Artificial Intelligence or AI has turned digital content production on its head. From software that writes copy to image creatives, AI is breaking innovation barriers. Of the most mind-boggling applications is perhaps in the realm of text-to-image generation—a place where AI models take the textual descriptions and convert them into one-of-a-kind visuals.

ImagiGen AI is a cutting-edge SaaS tool that focuses on making the AI image generation process easy for end users. Unlike traditional tools that need extensive technical expertise or command-line interaction, ImagiGen AI has a user-friendly and straightforward interface. The user types in a prompt, e.g., "a futuristic city at night," and the application runs it through the Clipdrop API to create a resultant image.

With features such as user login integrated into it, a credit-based usage system, image storage and management, and a secure payment gateway, the platform provides a seamless experience for both layman and professional users.

1.1 SCOPE OF THE PROJECT

1.1.1 Existing System:

Over the last couple of years, the domain of AI-generated material has been headed by some popular tools such as DALL·E (created by OpenAI), MidJourney, and Stable Diffusion (created by Stability AI). These tools have the ability to generate high-fidelity and good-quality images out of natural language inputs. Even though they are huge technological gains, they're not without problems—particularly with regard to an average or even non-technical user.

Some of the biggest disadvantages of the current systems are:

- Integrate Complex Interfaces: Most of them work through Discord bots, command-line interfaces, or developer-cantered APIs, thereby excluding users lacking technical skills. For instance, creating an image in Midjourney demands possession of a Discord account and acquaintance with chat commands, which makes it disconcerting for beginners.

- High Access Cost: Platforms tend to employ costly subscription plans. Such plans can provide a finite number of generations per month, and plan upgrades can be pricey, particularly for small creators, freelancers, or students.
- Absence of End-to-End Features: Though image generation is the primary emphasis, other features like image management, secure user authentication, payment integration, and user-Friendly dashboards are typically missing or not well-implemented.
- Limited Customization: The majority of tools currently available do not permit straightforward tracking of image history or asset storage for later access, and they may not support account-based usage control or credit-based systems.

As a result of these limitations, there is room for a more user-focused platform that serves novice and professional users alike without unnecessary complexity.

1.1.2 Problem Statement:

Although AI image generation technology is quickly evolving, a huge gap still exists to bring this technology into the reach of affordability and usability. The majority of the current solutions serve large organizations or tech-savvy users, and the large part of the market is left behind—everyday users, artists, small companies, teachers, and content creators who require a more user-friendly, more manageable solution.

A platform with a growing demand is one that:

- Provides easy-to-use functionality without demanding technical expertise.
- Offers an economical means of access to AI-produced content.
- Supports key functionality like user account management, picture storage, and adaptable payment arrangements.
- Facilitates personalization and history recording for reuse and creative experimentation.

The absence of such platforms provides a hindrance to the deployment of AI aids in day-to-day creative workflows.

1.1.3 Proposed System:

ImagiGen AI is created to solve these issues through a complete, easy-to-use, and cost-effective SaaS solution for AI-driven image creation. It is built to unlock a broad variety of users—ranging from digital creators and influencers to startups and design teams—by blending AI innovation with functional web-based capabilities.

Key features of the system in question are:

- Clean, Intuitive UI/UX: Minimalistic and responsive web interface that streamlines the process of generating images to a few clicks—login, input a prompt, and generate.
- Credit-Based Usage System: Credits are assigned to users, which are used as each image is generated. This offers a flexible pay-as-you-go system, assisting users with their budget and usage planning.
- Secure Payment Gateway Integration: The site integrates with reliable payment gateways such as Stripe or Razorpay, enabling users to buy credits securely using popular payment options.
- Image Storage: Each image created is saved in the user's profile, along with its prompt, date, and re-download or delete options. This feature assists in keeping a creative archive and enables future reference.
- Scalability and Modularity: Designed on the MERN stack and modular API integration, the system can be scaled without difficulty to manage higher user load

2. SYSTEM ANALYSIS

2.1 Review of literature:

Recent breakthroughs in Artificial Intelligence, especially in Natural Language Processing (NLP) and Computer Vision, have seen machines convert text prompts into realistic-looking images. Platforms such as DALL·E, Stable Diffusion, and MidJourney have been powered by technologies such as GANs, Diffusion Models, and transformer-based architectures.

Studies indicate that SaaS-based AI solutions provide easier access to complicated technologies by means of web platforms, particularly when combined with easy-to-use designs and open payment systems such as credit models. Research further highlights that including secure payment channels and providing ease-of-use UI/UX tremendously boosts user involvement and confidence.

ImagiGen AI takes advantage of these innovations, providing a simplified, inexpensive, and convenient platform for image generation using AI.

2.2 Feasibility study:

- **Technical feasibility:**

ImagiGen AI is technically feasible with the use of modern and reliable technologies. The MERN stack (MongoDB, Express.js, React.js, Node.js) provides a strong foundation for developing scalable web applications. Integration with the Clipdrop API ensures high-quality AI image generation without the need to build complex models from scratch. Secure payment gateways like Stripe or Razorpay can be easily integrated using available SDKs.

- **Economic feasibility:**

ImagiGen AI is financially viable given the low costs of development and operation that come with employing open-source technologies such as the MERN stack. Cloud hosting fees, Clipdrop API fees, and payment gateway fees are reasonable within a tight budget.

- **Behavioural feasibility:**

The system is behaviourally viable because it responds to actual user needs—providing an easy and accessible method for creating AI images without technical expertise. B

- **Schedule feasibility**:

The timeline for developing ImagiGen AI is achievable and reasonable. Since the MERN stack has a modular framework, processes such as authentication, image creation, payment gateway integration, and UI development can be executed in parallel.

2.3 Tools technology:

FRONT-END	React.js, HTML5, CSS3, Tailwind CSS and JavaScript
BACK-END	Node.js, Express.js, MongoDB and Mongoose
TOOLS	Visual Studio Code, Postman , Git & GitHub, Figma and DB Designe
OPERATING SYSTEM	Linux or MacOS or Microsoft Windows
Third-Party APIs	Clip drop API and Razor pay API

2.4 Hardware requirements:

CPU	INTEL CORE 2 DUO 2.0 GHz
RAM	2.00 GB
Hard disk	50 GB
Graphics	Integrated Graphics card

2.4 Software Requirements:

Front End	React.js, HTML5, CSS3, JavaScript (ES6+), Tailwind CSS
Back End	Node.js with Express.js
Database	MongoDB with Mongoose
APIs Integrated	Clipdrop API (AI Image Generation), Stripe / Razorpay (Payment Gateway), NodeMailer (Email Notifications)
Development Tools	Visual Studio Code, Postman, GitHub, Figma, dbDesigner
Operating System	Linux (Ubuntu), macOS, Microsoft Windows

3. System Design

3.1 Modules in the software:

- **Authentication Module**
  Handles user sign-up, login, password encryption, and session management.
- **Image Generation Module**
  Accepts text prompts and generates images by interacting with the Clipdrop API.
- **Credit Management Module**
  Monitors credit usage, deducts credits per image generation, and displays current balance.
- **Payment Module**
  Manages secure transactions and credit purchases using integrated payment gateways.
- **User Dashboard Module**
  Displays generated image history, credit details, and allows downloads.

3.2 Functional and Non- functional requirements

3.2.1 Functional Requirements

Functional requirements define the essential operations and services that ImagiGen AI must provide to ensure a smooth and efficient user experience. These requirements specify how the system responds to various inputs and user interactions.

- **User Registration and Login:**
  The system shall allow users to create accounts and authenticate securely using their email and password. All login sessions will be managed using JSON Web Tokens (JWT).
- **Text-to-Image Generation:**
  The system shall accept user-submitted text prompts and generate corresponding images using the Clipdrop API. Each generation will deduct one credit from the user's available balance.
- **Credit Management**

The application shall maintain a credit-based system where users can check available credits, purchase more through payment gateways (e.g., Stripe/Razorpay), and receive OTP-based verification for secure transactions.

- **Payment Integration**

  The platform shall provide a secure payment interface for credit purchases. The system will confirm transactions using OTP verification before updating the user's credit balance.

- **Error Handling and Feedback**

  The application shall validate inputs and provide clear error messages in case of invalid prompts, failed payments, or insufficient credits, ensuring users are guided effectively during operations.

3.2.2 Non-Functional Requirements

Non-functional requirements define the quality attributes, performance benchmarks, and system constraints that guide how ImagiGen AI operates. These requirements ensure the application delivers a robust, scalable, and user-centered experience.

- **Usability**

  ImagiGen AI shall provide a simple, clean, and intuitive user interface. Users, regardless of technical background, should be able to register, input prompts, generate images, and manage their image history with minimal learning curve.

- **Responsiveness**

  The application must function smoothly across various devices—desktops, tablets, and smartphones. The UI components shall adjust dynamically to different screen sizes to provide a consistent user experience.

- **Data Integrity and Security**

  All user data including login credentials, credit balances, and generated images must be securely stored. The system shall ensure encrypted communication, secure payment processing, and role-based access controls where applicable.

- **Performance**

Image generation requests should be processed quickly via Clipdrop API, with minimal lag. Dashboard interactions (e.g., prompt submission, viewing history) must respond within acceptable time limits (ideally under 2 seconds).

- **Browser Compatibility**

  ImagiGen AI shall work consistently across all modern browsers including Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari, ensuring no functionality or layout breaks occur due to browser differences**.**

- **Scalability and Maintainability**

  The application shall be designed with modular architecture using the MERN stack. Future enhancements—such as additional image filters, AI model integration, or advanced admin controls—should be possible without reworking the core system.

3.3 System perspective:

ImagiGen AI is a web-based, independent SaaS application developed with the MERN stack (MongoDB, Express.js, React.js, and Node.js). It has a modular and scalable architecture that provides flexibility, maintainability, and high performance.

The system integrates with:

Clipdrop API for creating AI images from text prompts entered by users.

Stripe or Razorpay for secure payment with OTP-based verification.

User authentication is taken care of by secure login and token-based sessions. All the information, such as user data, image history, and credit balance, is stored efficiently using MongoDB. The front-end is built using React.js and styled using Tailwind CSS, making it responsive across devices.

ImagiGen AI is made to provide a glitch-free and user-friendly experience, providing creative image generation feature without needing to have technical skills.

3.4 DFD diagrams:

A Data Flow Diagram (DFD) is a graphical representation of the flow of data within a system. It helps in understanding how data moves from input to processing and finally to

output or storage. In this project, the DFD illustrates how the Task Management System handles user interactions and manages task data.

3.4.1 DFD Level 0:

Imagine you want a picture of a fluffy cloud shaped like a cat. You tell this to a friendly artist, our "ImagiGen AI System 0". Now, this artist, while talented, sometimes needs a little extra help with the really imaginative stuff. So, they reach out to a super-genius art assistant, the "OpenAI API", and describe your cloud-cat idea in detail. The super-genius assistant gets to work and paints your whimsical cloud-cat. Once it's done, they send the artwork back to our friendly artist. Finally, the artist proudly presents your unique "Generated Image" of the cloud-cat right to you!

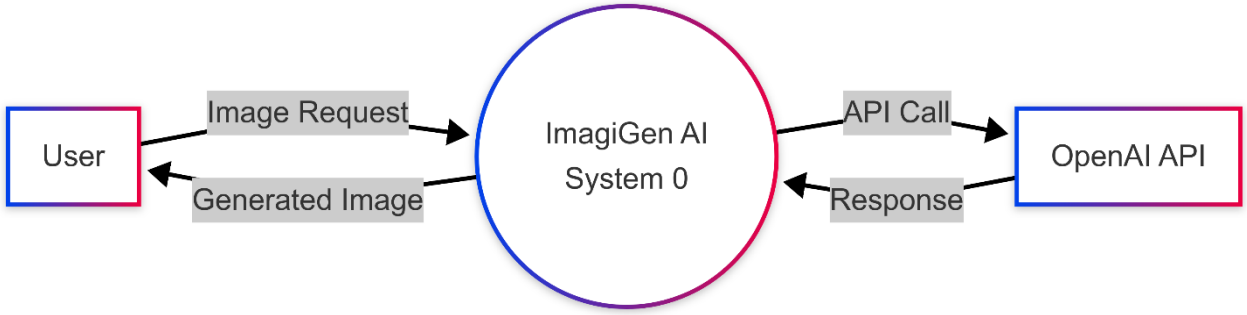


Fig1: Level-0 DFD

3.4.2 DFD Level 1:

This diagram illustrates an image generation system that incorporates user authentication. Initially, a "Client" interacts with the system by attempting to "Login/Register". This action triggers "Authentication Process 1", which verifies the user's credentials against "User Data" stored in a "MongoDB" database. The authentication process retrieves "Stored Data" from MongoDB to perform this verification. Upon successful authentication, the system issues an "Auth Token" to the client. Subsequently, the client can initiate the "Image Generation Process 2", which makes an "API Call" to the "OpenAI API", likely including the user's

image request and the "Auth Token". The "OpenAI API" then generates the requested image and sends it back as a "Response" to the "Image Generation Process 2". Finally, the generated image is delivered back to the authenticated "Client". This design ensures that only logged-in users can access the image generation capabilities provided by the integration with the OpenAI API.

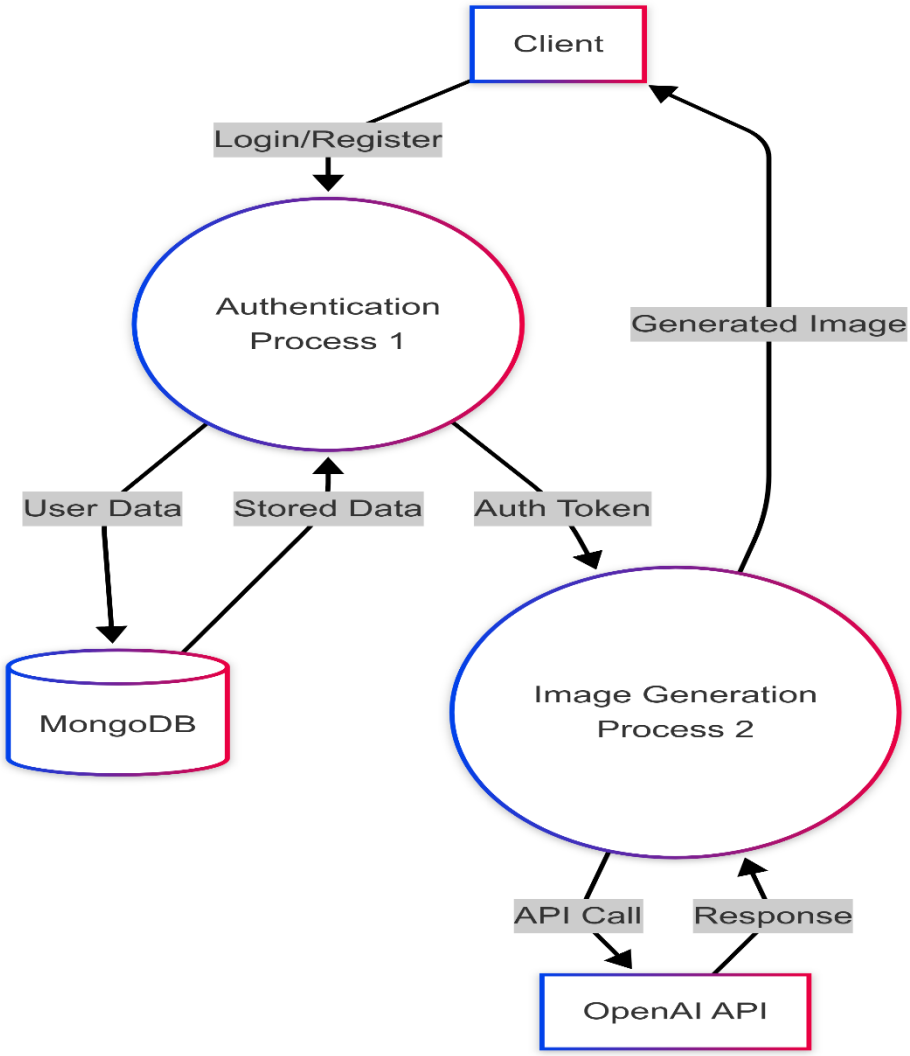


Fig2: Level-1 DFD

3.4.3 DFD Level 2:

The diagram outlines an authenticated image generation flow. A "Client Browser" sends "Login Data" to "Authentication 1.0", which verifies the user against "User Records" in a "User Database". Upon successful verification, a "Token" is issued. For image generation, the "Client Browser" sends a request along with the "Token" to "Auth Middleware 1.1",

which validates the token. If "Validated", the request proceeds to "Image Generation 2.0", which sends an "API Request" to the "OpenAI API" and receives an "AI Response" (the "Generated Image") to be sent back to the "Client Browser".

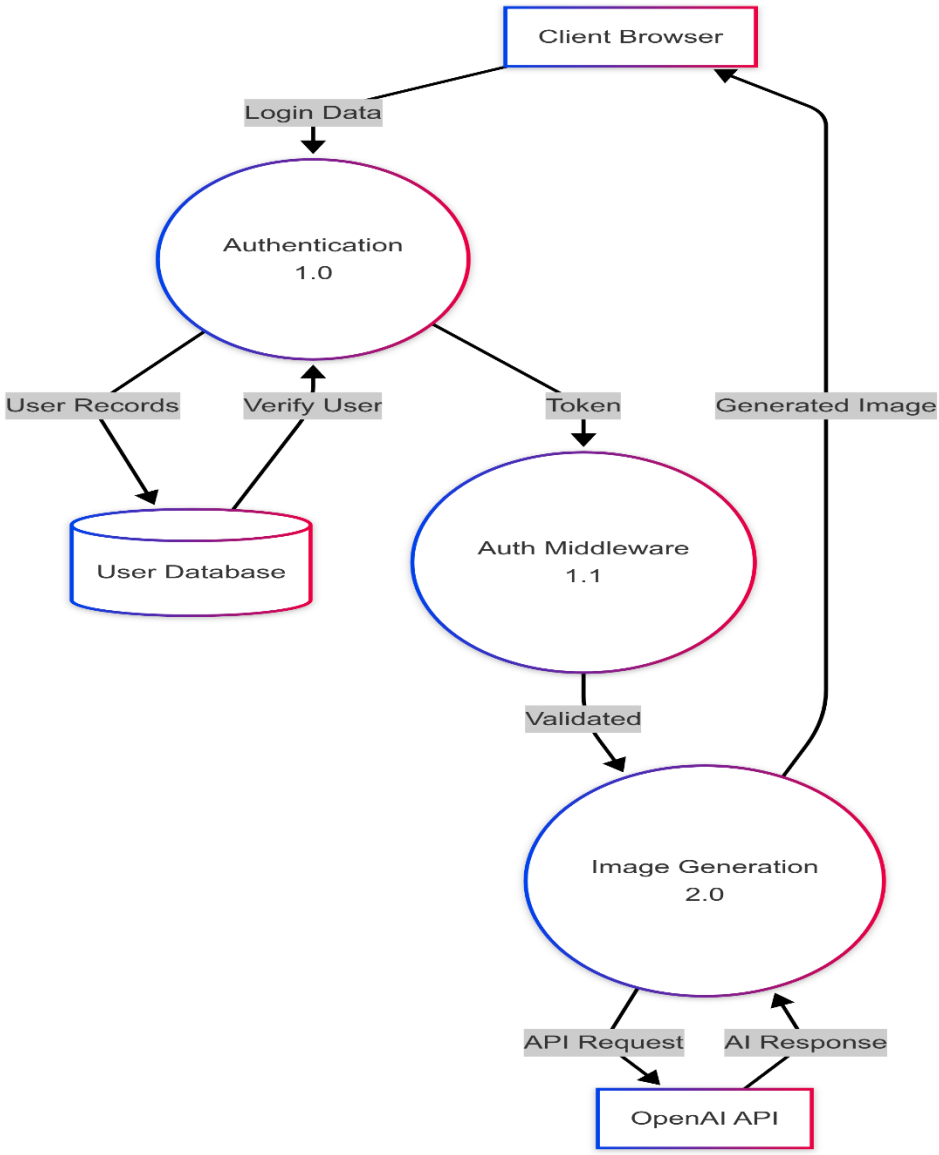


Fig3: Level-2 DFD

3.5 UML Diagrams:

3.5.1 Use Case Diagram

The Use Case Diagram of the Task Management System represents the different functionalities provided to two main actors: Admin and User. The Admin has control over creating, editing, deleting, and assigning tasks to users. The Admin can also view all tasks, manage user assignments, view timesheet reports, and generate productivity reports. On the

other hand, the User can view their assigned tasks, update the status of their tasks, view detailed information about tasks, see a calendar view of their schedules, and manage timesheets by submitting and editing their work hours.

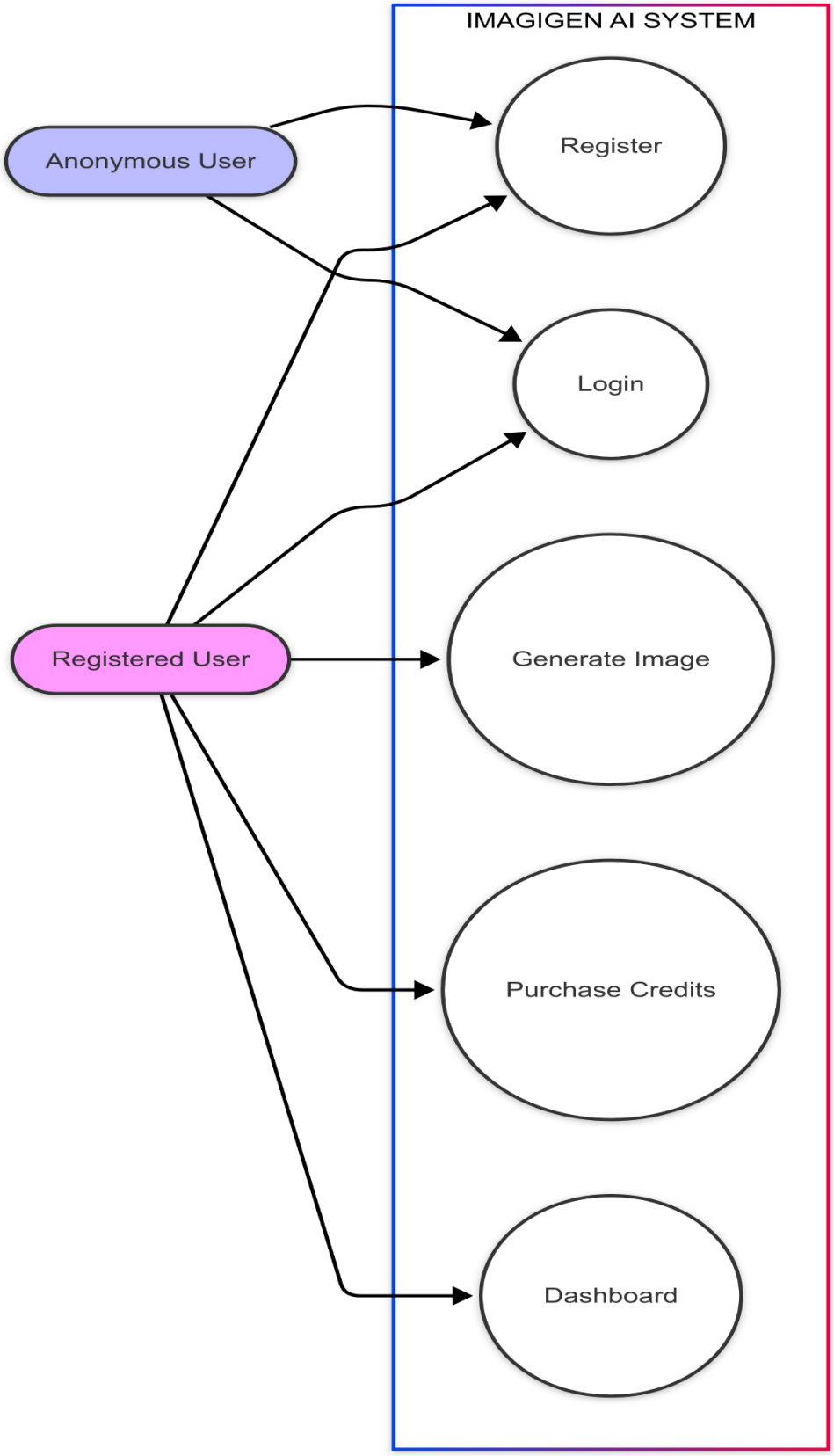


Fig4: Use Case Diagram

3.5.2 Activity Diagram

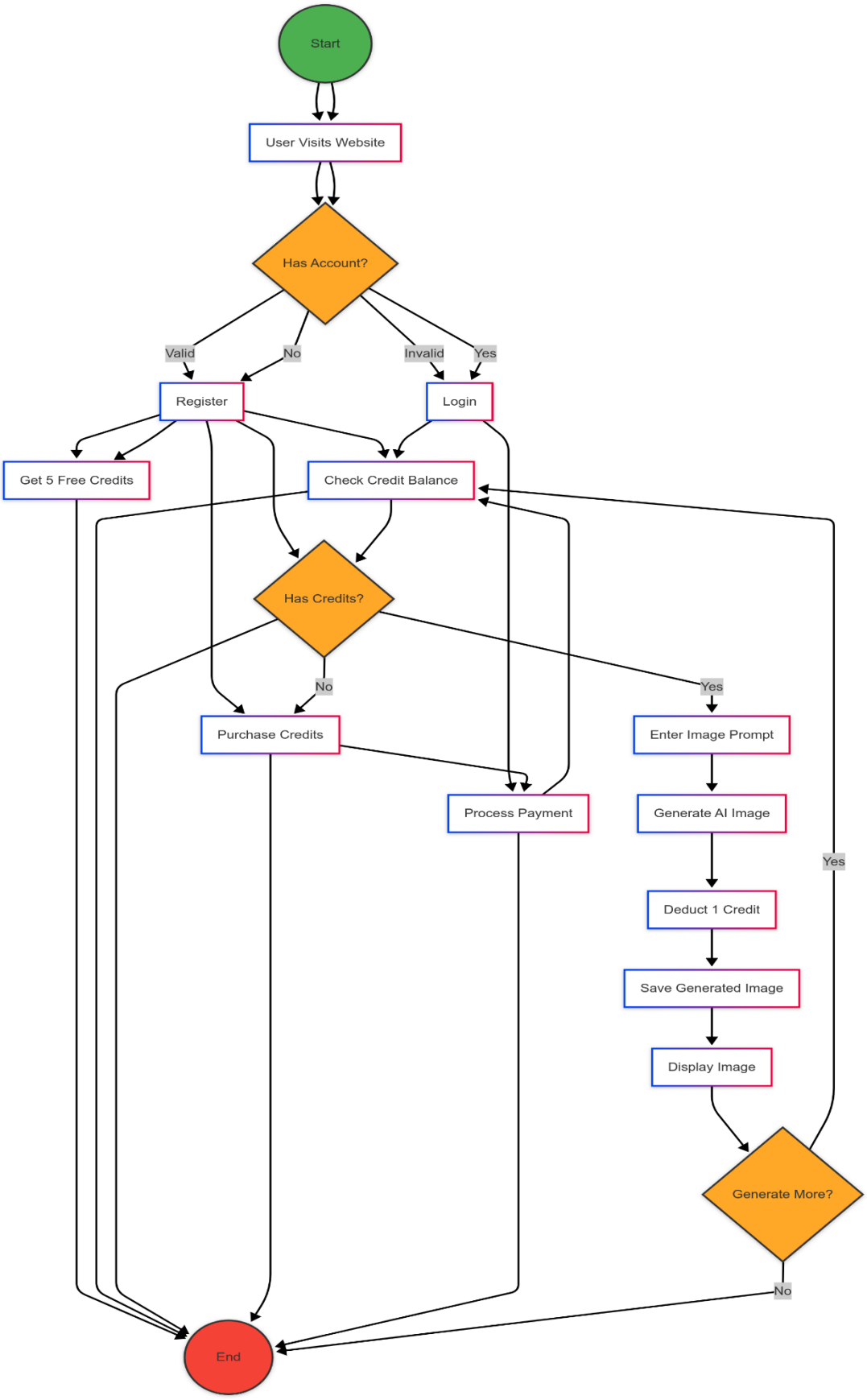


Fig5: Activity Diagram

3.5.3 Class Diagram:

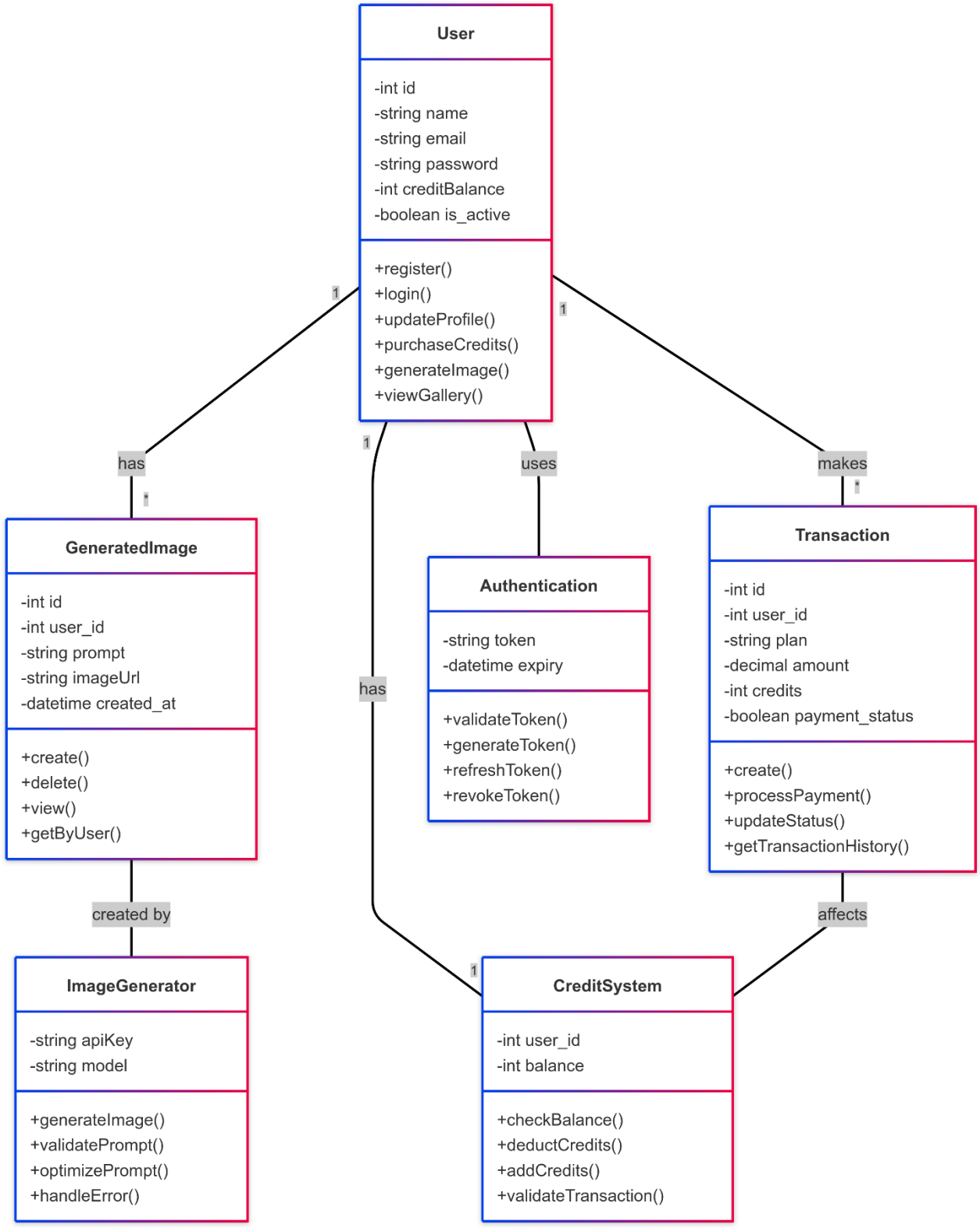


Fig6: Class Diagram

3.5.4 Sequence Diagram:

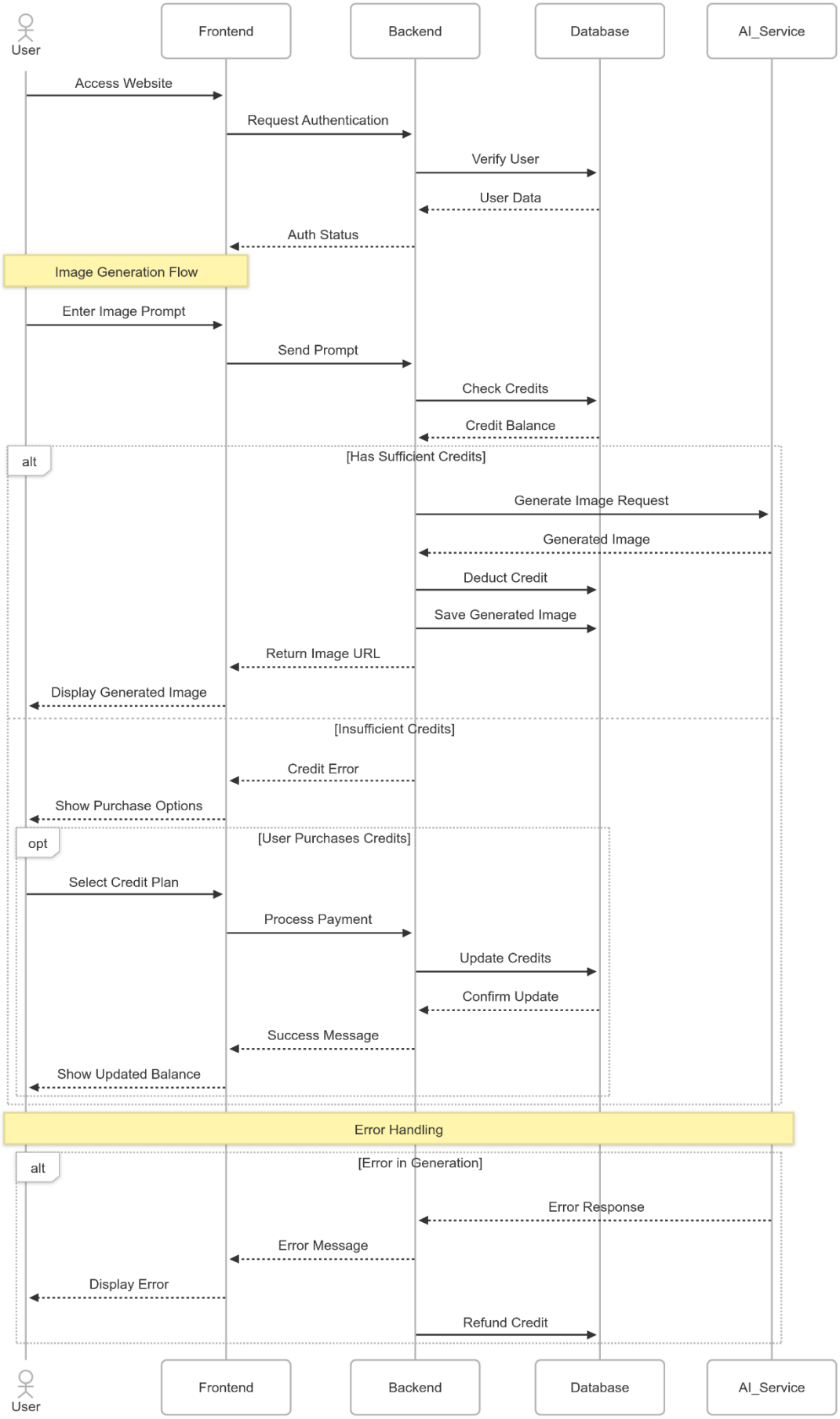


Fig7: Sequence Diagram

3.5.5 ER Diagram

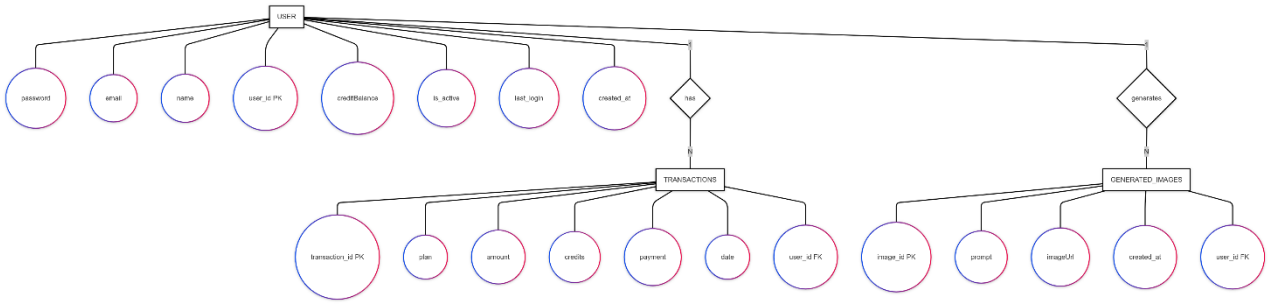
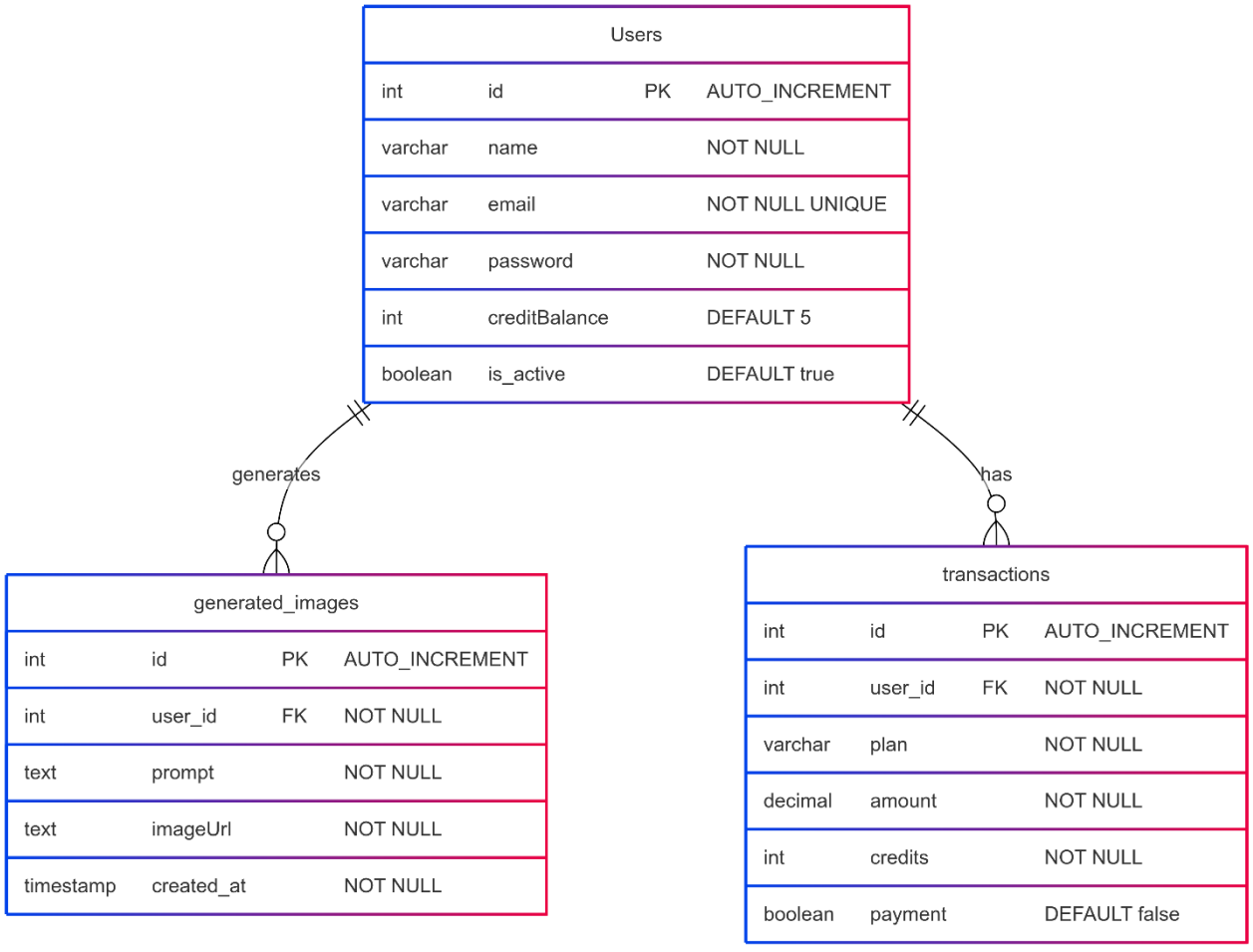


Fig8: ER Diagram

3.5.6 Database Diagram:



3.7 Payment Transaction structure:

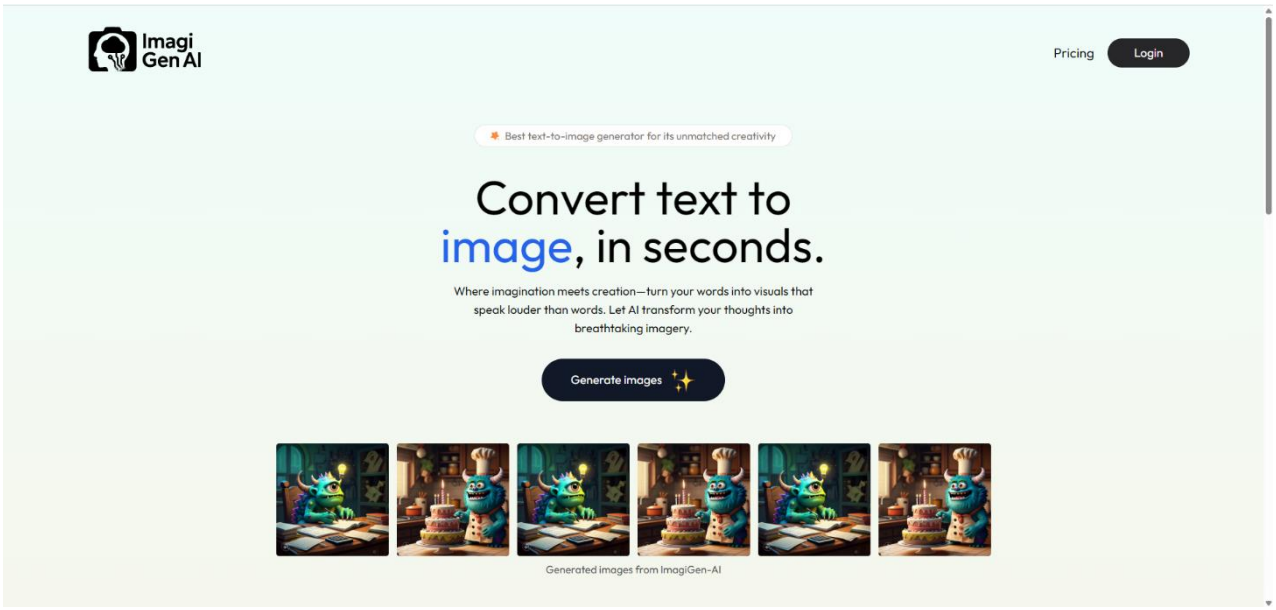

```
_id: ObjectId('680b56e372666ce0106edac1')
userId : "68022bf778b51438cf2ca131"
plan : "Business"
amount : 250
credits : 5000
payment : true
date : 1745573603248
__v : 0
```

```
_id: ObjectId('680b56e372666ce0106edac4')
userId : "68022bf778b51438cf2ca131"
plan : "Business"
amount : 250
credits : 5000
payment : true
date : 1745573603411
__v : 0
```

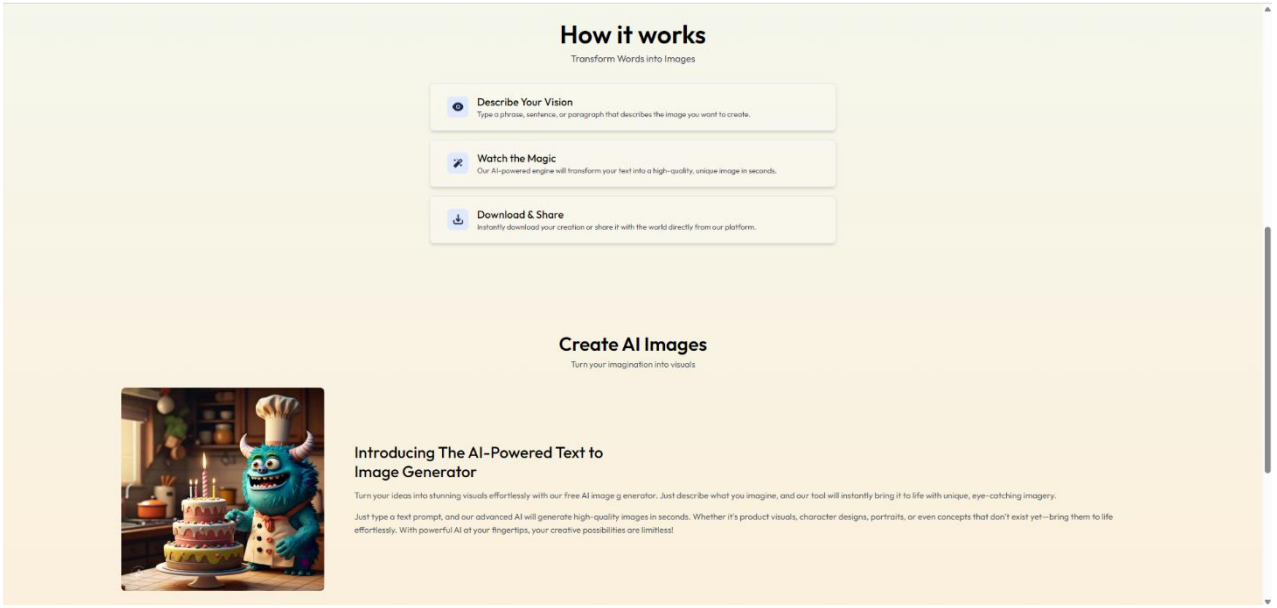
```
_id: ObjectId('680b56e372666ce0106edac7')
userId : "68022bf778b51438cf2ca131"
plan : "Business"
amount : 250
credits : 5000
payment : true
date : 1745573603586
__v : 0
```

```
_id: ObjectId('680b5711872e4ff992aaceeb')
userId : "68022bf778b51438cf2ca131"
plan : "Basic"
amount : 10
credits : 100
payment : true
date : 1745573649804
__v : 0
```

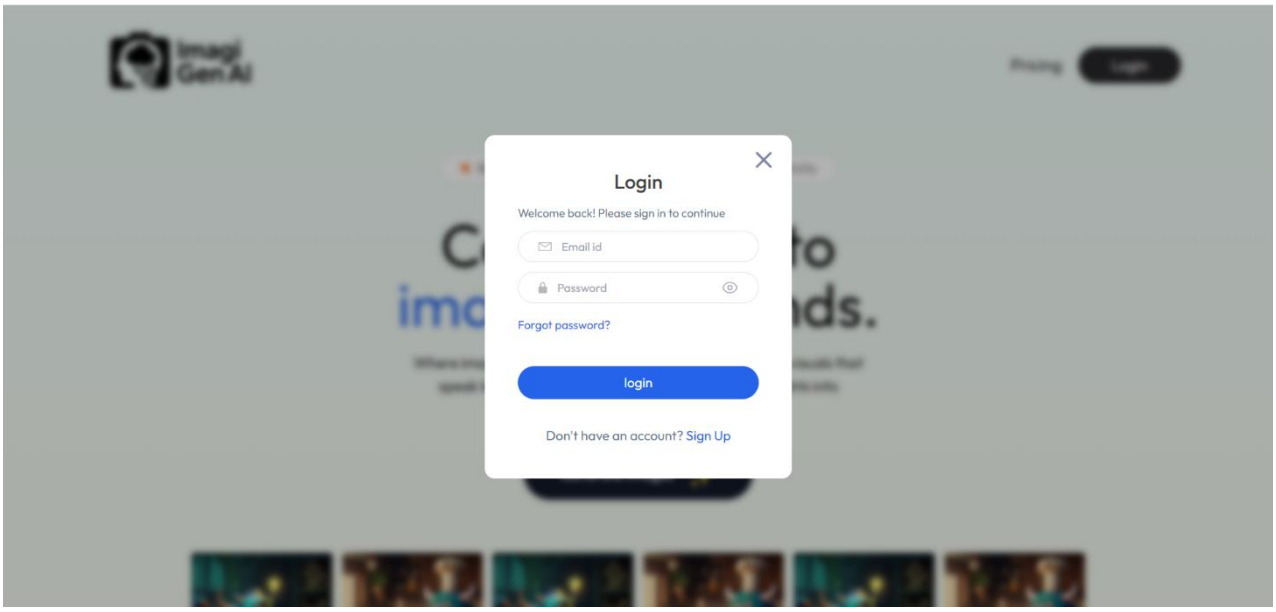
4. Screen shots:



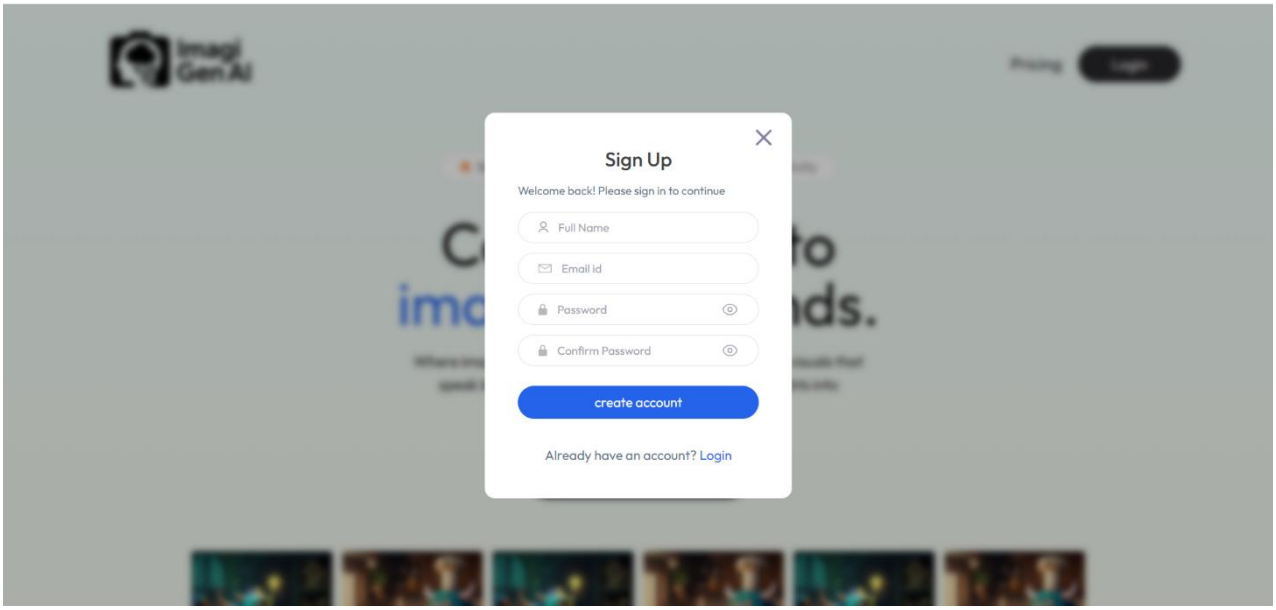
Main page start



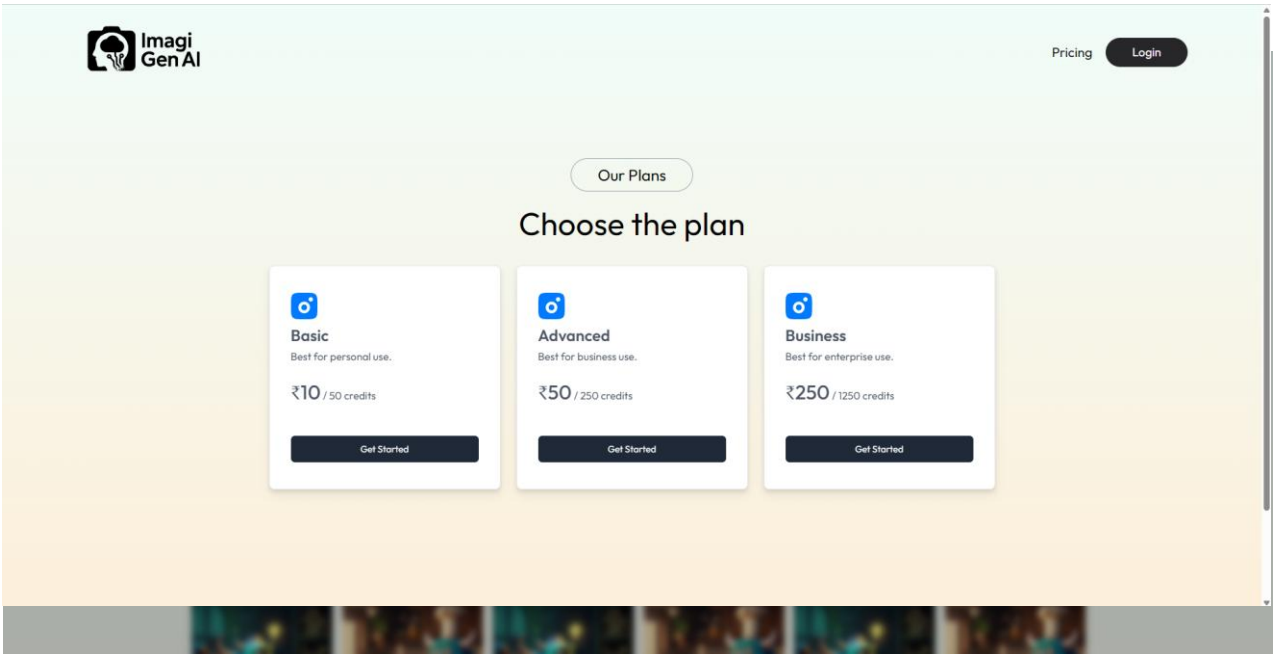
Main page end



Login



Sign Up



Reset Password

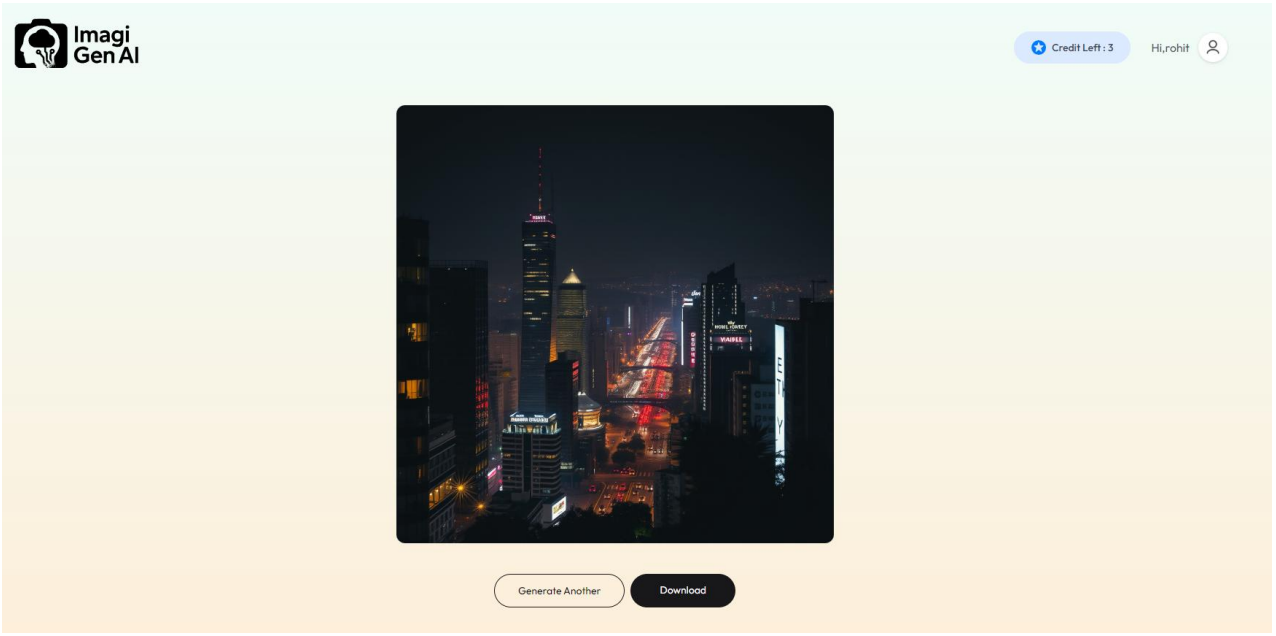
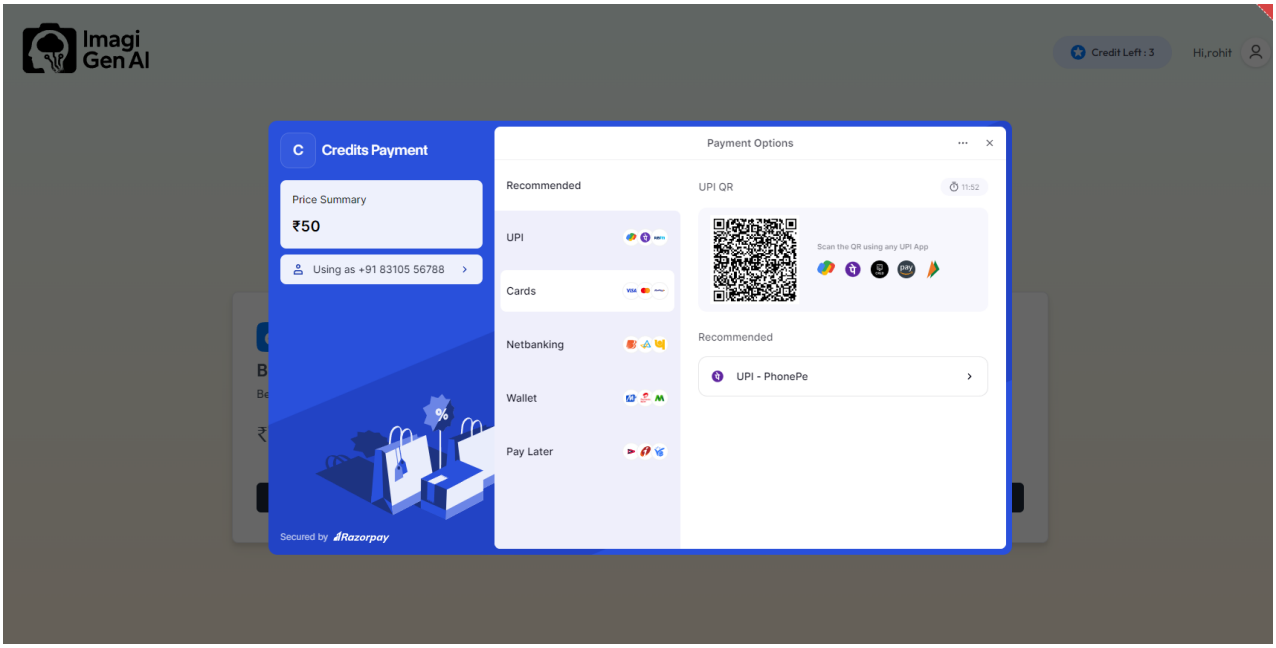


Image generation



Pricing and payment

5. Software testing

5.1 Test cases

5.1.1 Functional Testing:

Functional testing is an important step in the software testing process that deals with the verification of the functional features of a software application. It verifies that the software operates as per the defined requirements and executes all the intended functions correctly. Functional testing mainly deals with testing individual functions, user commands, data manipulation, and business processes to ensure that the software system works as per the specified specifications.

The objective of functional testing is to ensure that every feature of the
software behaves according to the requirements document. It includes checking user interfaces, APIs, databases, security functionalities, client/server communications, and other features. Testers emulate real system use and ensure that the system's response to a range of input scenarios produces the expected output.

Functional testing is typically done through manual testing or automated software and adheres to methods like black box testing. Functional testing types are unit testing, integration testing, system testing, regression testing, and user acceptance testing. For instance, in a login module, functional testing would verify that valid users can log in, invalid users cannot log in with the wrong information, and proper error messages are shown when blank required fields are present. functional testing really improves software quality, user satisfaction, and system reliability.

Test Case ID	Feature	Test Scenario	Expected Result	Status
TC-F001	Registration	Register with valid details	Account created successfully	Pass
TC-F002	Registration	Register with existing email	Error: Email already exists	Fail
TC-F003	Login	Valid credentials	Redirect to dashboard	Pass
TC-F004	Login	Wrong password	Error: Invalid credentials	Pass
TC-F005	Auth	Access dashboard without login	Redirect to login page	Fail
TC-F006	Prompt	Submit valid prompt	Image generated	Pass
TC-F007	Prompt	Submit empty prompt	Error shown: Prompt required	Pass
TC-F008	Credit System	Sufficient credits	Image generated, credit deducted	Pass
TC-F009	Credit System	No credits	Error: Insufficient credits	Fail
TC-F010	Payment	Valid card payment	Credits added	Pass
TC-F011	Payment	Cancel payment	No credits added	Pass
TC-F012	Download	Download generated image	Image saved	Pass
TC-F013	Logout	Logout from account	Redirected to login	Pass

5.1.2 Non-Functional Testing:

Non-functional testing plays a similar vital role in the software testing lifecycle as it considers elements of the system that relate to non-specific behaviours or functionalities. Instead of concentrating on what the software can do, non-functional testing targets how the system behaves under several conditions. Non-functional testing focuses on ensuring performance, usability, reliability, scalability, and other non-functional traits of the software meet expectations.

Such a test is aimed at ensuring that the system operates as expected extensive loads, recovers from stress and peak conditions well, offers an easy-to-use and seamless user interface, preserves data integrity during crash recovery, and supports safe interactions. For example, it checks how fast a system responds to a user's actions, how many users can use the system at a given time, how fast the system recovers after a crash, and whether or not it is accessible on various devices and browsers.

Non-functional testing comprises different subtypes like performance
testing, load testing, stress testing, usability testing, compatibility testing, maintainability testing, and security testing. Although frequently neglected in initial phases, these tests are crucial for long-term system stability, scalability, and user retention. A properly running system without performance optimization or under stress could seriously affect the user experience and business reputation.

Test Case ID	Aspect	Test Scenario	Expected Result	Status
TC-NF001	Performance	Load app with 1000 users	Stable performance	Pass
TC-NF002	Security	Try SQL injection	Input blocked/sanitized	Pass
TC-NF003	Responsiveness	Open on mobile & tablet	Layout adjusts well	Pass
TC-NF004	Session	Idle for 30 mins	Session expired	Pass
TC-NF005	Compatibility	Chrome, Firefox, Edge	Same behavior	Fail
TC-NF006	API Error	Clipdrop API fails	Error message displayed	Pass
TC-NF007	Recovery	Browser crash during generation	Session resumes	Fail
TC-NF008	Uptime	24-hour run	App remains online	Pass
TC-NF009	Accessibility	Keyboard-only navigation	Fully operable	Pass
TC-NF010	Usability	New user trial	Easy navigation	Pass

6. Sample Coding

API's:

VITE_BACKEND_URL= 'http://localhost:4000'

VITE_RAZORPAY_KEY_ID = "rzp_test_YynlYCEl95bkg7"

MONGODB_URI
="mongodb+srv://rohitdesai:rohit713@cluster0.pmmvfwg.mongodb.net"

JWT_SECRET= "secret#text"

CLIPDROP_API='80bcba45574e8db780d20f6a7b872220ea3c5a320d3a338ecb1
3e086a4e04252b9f575b33b3ce0d4e3b37dc0fe26ec49'

RAZORPAY_KEY_ID = "rzp_test_YynlYCEl95bkg7"

RAZORPAY_KEY_SECRET = "CPoG8ub0I983YqjUGily6WM9"

CURRENCY = "INR"

```
import React, { useContext } from 'react'
import { Routes, Route } from 'react-router-dom'
import Home from './pages/Home'
import Result from './pages/Result'
import Navbar from './components/Navbar'
import BuyCredit from './pages/BuyCredit'
import Footer from './components/Footer'
import Login from './components/Login'
import ForgotPassword from './components/ForgotPassword'
import { AppContext } from './context/AppContext'
import { ToastContainer} from 'react-toastify';
import 'react-toastify/dist/ReactToastify.css';
```

```
const App = () => {
  const {showLogin, showForgotPassword} = useContext(AppContext)
  return (
    <div className='px-4 sm:px-10 md:px-14 lg:px-28 min-h-screen bg-gradient-to-b from-teal-50 to-orange-100'  >
      <ToastContainer position='bottom-right' />
    <Navbar/>
    {showLogin && <Login/>}
    {showForgotPassword && <ForgotPassword/>}
      <Routes>
        <Route path='/' element={<Home />} />
        <Route path='/result' element={<Result />} />
        <Route path='/buy' element={<BuyCredit />} />
      </Routes>
    <Footer/>
    </div>
  )
}

export default App

import React, { useState, useContext, useEffect } from 'react';
import { assets } from '../assets/assets';
import { AppContext } from '../context/AppContext';
import { motion } from 'framer-motion';
import axios from 'axios';
import { toast } from 'react-toastify';
```

```
const ForgotPassword = () => {
    const [email, setEmail] = useState('');
    const [newPassword, setNewPassword] = useState('');
        const { backendUrl, setShowLogin, setShowForgotPassword } = useContext(AppContext);

    const handleSubmit = async (e) => {
        e.preventDefault();
        try {
            const { data } = await axios.post(`${backendUrl}/api/user/reset-password`, {
                email,
                newPassword
            });

            if (data.success) {
                toast.success('Password successfully changed!');
                setShowForgotPassword(false);
                setShowLogin(true);
            } else {
                toast.error(data.message);
            }
        } catch (error) {
            toast.error(error.response?.data?.message || error.message);
        }
    };
```

```
    useEffect(() => {
        document.body.style.overflow = 'hidden';
        return () => {
            document.body.style.overflow = 'unset';
        }
    }, []);


    return (
        <div className='fixed top-0 left-0 right-0 bottom-0 z-10 backdrop-blur-sm bg-black/30 flex justify-center items-center'>
            <motion.form
                onSubmit={handleSubmit}
                initial={{ opacity: 0.2, y: 100 }}
                transition={{ duration: 0.8 }}
                whileInView={{ opacity: 1, y: 0 }}
                viewport={{ once: true }}
                className='relative bg-white p-10 rounded-xl text-slate-500'
            >
                        <h1 className='text-center text-2xl text-neutral-700 font-medium'>Reset Password</h1>
                <p className='text-sm mt-3 text-center'>Enter your email and new password</p>


            <div className='border px-6 py-2 flex items-center gap-3 rounded-full mt-6'>
                <img src={assets.email_icon} alt="" />
                <input
                    onChange={(e) => setEmail(e.target.value)}
```

```
                value={email}
                type="email"
                className='outline-none text-sm w-full'
                placeholder='Email address'
                required
            />
        </div>

        <div className='border px-6 py-2 flex items-center gap-3 rounded-full
mt-3'>
            <img src={assets.lock_icon} alt="" />
            <input
                onChange={(e) => setNewPassword(e.target.value)}
                value={newPassword}
                type="password"
                className='outline-none text-sm w-full'
                placeholder='New password'
                required
            />
        </div>

          <button className='bg-blue-600 w-full text-white py-2 rounded-full
mt-6'>
            Reset Password
        </button>

        <p className='mt-5 text-center'>
            Remember your password?{' '}
```

```
                <span
                  className='text-blue-600 cursor-pointer'
                  onClick={() => {
                    setShowForgotPassword(false);
                    setShowLogin(true);
                  }}
                >
                  Login
                </span>
              </p>
              <img
                onClick={() => setShowForgotPassword(false)}
                src={assets.cross_icon}
                alt=""
                className='absolute top-5 right-5 cursor-pointer h-5 w-7 opacity-75'
              />
            </motion.form>
          </div>
    );
};

export default ForgotPassword;

// logo and login purchas
import React, { useContext } from 'react'
import { assets } from '../assets/assets'
import { Link, useNavigate } from 'react-router-dom'
```

```
import { AppContext } from '../context/AppContext'


const Navbar = () => {
   const {user,setShowLogin,logout,credit}=useContext(AppContext)
   const navigate = useNavigate()
   return (
      <div className='flex items-center justify-between  -mt-8'>
         <Link to='/'>
            <img src={assets.logo} alt="" className='w-28 sm:w-32 lg:w-52 ' />
         </Link>


         <div>
            {
               user ?
                  <div className='flex items-center gap-2 sm:gap-3'>
                   <button onClick={() => navigate('/buy')} className='flex items-center gap-2 bg-blue-100 px-4 sm:px-6 py-1.5 sm:py-3 rounded-full hover:scale-105 transition-all duration-500'>
                        <img className='w-5' src={assets.credit_star} alt="" />
                         <p className='text-sm sm:text-sm font-medium text-gray-700 '>Credit Left : {credit}</p>
                    </button>
                              <p  className='text-gray-600  max-small:hidden pl-4'>Hi,{user.name} </p>
                  <div className='relative group'> <img src={assets.profile_icon} className='w-10 drop-shadow' alt="" />
                        <div className='absolute hidden group-hover:block top-0 r-0 z-10 text-black rounded pt-12'>
```

```
                        <ul className='list-non m-0 p-2 bg-blue-100 rounded-md
border text-sm'>
                            <li onClick ={logout}  className='py-1 px-2 cursor-
pointer pr-10'>
                            Logout
                          </li>
                        </ul>
                      </div>
                    </div>
                  </div>

                  :
                  <div className='flex items-center gap-2 sm:gap-5'>

                    <p onClick={() => navigate('/buy')} className='cursor-pointer
text-xl'>Pricing</p>
                    <button onClick={()=>setShowLogin(true)} className='bg bg-
zinc-800 text-white px-10 py-2 sm:py-2 text-lg rounded-full' >Login</button>
                  </div>
              }
            </div>
        </div>
    )
}

export default Navbar

import React, { useContext, useEffect, useState } from 'react'
```

```
import { assets } from '../assets/assets'
import { AppContext } from '../context/AppContext'
import { motion } from 'framer-motion'
import axios from 'axios'
import { toast } from 'react-toastify'
import { useNavigate } from 'react-router-dom'
import { EyeIcon, EyeSlashIcon } from '@heroicons/react/24/outline'

const validateEmail = (email) => {
    return String(email)
        .toLowerCase()
        .match(/^[^\s@]+@[^\s@]+\.[^\s@]+$/)
}

const validatePassword = (password) => {
    return password.length >= 8
}

const Login = () => {
    const [state, setState] = useState('Login')
    const { setShowLogin, setShowForgotPassword, backendUrl, setToken, setUser
} = useContext(AppContext)
    const navigate = useNavigate()

    const [name, setName] = useState('')
    const [email, setEmail] = useState('')
    const [password, setPassword] = useState('')
    const [confirmPassword, setConfirmPassword] = useState('')
```

```
const [showPassword, setShowPassword] = useState(false)
const [showConfirmPassword, setShowConfirmPassword] = useState(false)

const [errors, setErrors] = useState({
    name: '',
    email: '',
    password: '',
    confirmPassword: ''
})

const validateForm = () => {
    const newErrors = {
        name: '',
        email: '',
        password: '',
        confirmPassword: ''
    }
    let isValid = true

    if (state === 'Sign Up' && !name.trim()) {
        newErrors.name = 'Name is required'
        isValid = false
    }

    if (!email.trim()) {
        newErrors.email = 'Email is required'
        isValid = false
```

```
    } else if (!validateEmail(email)) {
      newErrors.email = 'Invalid email format'
      isValid = false
    }

    if (!password) {
      newErrors.password = 'Password is required'
      isValid = false
    } else if (state === 'Sign Up' && !validatePassword(password)) {
      newErrors.password = 'Password must be at least 8 characters'
      isValid = false
    }

    if (state === 'Sign Up' && password !== confirmPassword) {
      newErrors.confirmPassword = 'Passwords do not match'
      isValid = false
    }

    setErrors(newErrors)
    return isValid
  }

  const onSubmitHandler = async (e) => {
    e.preventDefault()

    if (!validateForm()) {
      return
```

```
        }


        try {
            if (state === 'Login') {
              const { data } = await axios.post(`${backendUrl}/api/user/login`, { email,
password })
                if (data.success) {
                    setToken(data.token)
                    setUser(data.user)
                    localStorage.setItem('token', data.token)
                    setShowLogin(false)
                    toast.success('Logged in successfully')
                } else {
                    toast.error(data.message)
                }
            } else {
                 const { data } = await axios.post(`${backendUrl}/api/user/register`, {
name, email, password })
                if (data.success) {
                    setToken(data.token)
                    setUser(data.user)
                    localStorage.setItem('token', data.token)
                    setShowLogin(false)
                    toast.success('Account created successfully')
                } else {
                    toast.error(data.message)
                }
            }
```

```
        } catch (error) {
            toast.error(error.response?.data?.message || error.message)
        }
    }

    useEffect(() => {
        document.body.style.overflow = 'hidden'
        return () => {
            document.body.style.overflow = 'unset'
        }
    }, [])

    return (
        <div className='fixed top-0 left-0 right-0 bottom-0 z-10 backdrop-blur-sm bg-black/30 flex justify-center items-center'>
            <motion.form onSubmit={onSubmitHandler}
                initial={{ opacity: 0.2, y: 100 }}
                transition={{ duration: 0.8 }}
                whileInView={{ opacity: 1, y: 0 }}
                viewport={{ once: true }}
                className='relative bg-white p-10 rounded-xl text-slate-500 '>
                    <h1 className='text-center text-2xl text-neutral-700 font-medium '>{state}</h1>
                    <p className=' text-sm mt-3'>Welcome back! Please sign in to continue</p>

                {state !== 'Login' && (
                    <div className='flex flex-col gap-1'>
```

```
                    <div className={`border px-4 py-1 flex items-center gap-1 rounded-full mt-4 ${errors.name ? 'border-red-500' : name ? 'border-green-500' : ''}`}>
                    <img src={assets.profile_icon} className="w-7 h-7 opacity-50" alt="" />
                    <input
                        onChange={e => setName(e.target.value)}
                        value={name}
                        type="text"
                        className='outline-none text-sm w-full'
                        placeholder='Full Name'
                    />
                </div>
                    {errors.name && <span className='text-red-500 text-xs pl-4'>{errors.name}</span>}
            </div>
        )}
        <div className='flex flex-col gap-1'>
            <div className={`border px-6 py-2 flex items-center gap-3 rounded-full mt-3 ${errors.email ? 'border-red-500' : email && validateEmail(email) ? 'border-green-500' : ''}`}>
                <img src={assets.email_icon} alt="" />
                <input
                    onChange={e => setEmail(e.target.value)}
                    value={email}
                    type="email"
                    className='outline-none text-sm w-full'
                    placeholder='Email id'
                />
```

```
            </div>
                {errors.email && <span className='text-red-500 text-xs pl-6'>{errors.email}</span>}
            </div>
            <div className='flex flex-col gap-1'>
              <div className={`border px-6 py-2 flex items-center gap-3 rounded-full mt-3 ${errors.password ? 'border-red-500' : password && (state === 'Login' || validatePassword(password)) ? 'border-green-500' : ''}`}>
                <img src={assets.lock_icon} alt="" />
                <input
                  onChange={e => setPassword(e.target.value)}
                  value={password}
                  type={showPassword ? 'text' : 'password'}
                  className='outline-none text-sm w-full'
                  placeholder='Password'
                />
                <button
                  type="button"
                  onClick={() => setShowPassword(!showPassword)}
                  className="focus:outline-none"
                >
                  {showPassword ? (
                    <EyeSlashIcon className="h-5 w-5 text-gray-400" />
                  ) : (
                    <EyeIcon className="h-5 w-5 text-gray-400" />
                  )}
                </button>
              </div>
```

```
                {errors.password && <span className='text-red-500 text-xs pl-6'>{errors.password}</span>}
            </div>

            {state === 'Sign Up' && (
              <div className='flex flex-col gap-1'>
                <div className={`border px-6 py-2 flex items-center gap-3 rounded-full mt-3 ${errors.confirmPassword ? 'border-red-500' : confirmPassword && password === confirmPassword ? 'border-green-500' : ''}`}>
                  <img src={assets.lock_icon} alt="" />
                  <input
                    onChange={e => setConfirmPassword(e.target.value)}
                    value={confirmPassword}
                    type={showConfirmPassword ? 'text' : 'password'}
                    className='outline-none text-sm w-full'
                    placeholder='Confirm Password'
                  />
                  <button
                    type="button"
                    onClick={() => setShowConfirmPassword(!showConfirmPassword)}
                    className="focus:outline-none"
                  >
                    {showConfirmPassword ? (
                      <EyeSlashIcon className="h-5 w-5 text-gray-400" />
                    ) : (
                      <EyeIcon className="h-5 w-5 text-gray-400" />
                    )}
```

```
                    </button>
                </div>
                {errors.confirmPassword && <span className='text-red-500 text-xs pl-6'>{errors.confirmPassword}</span>}
            </div>
        )}

        {state === 'Login' && (
            <p onClick={() => {
                setShowLogin(false);
                setShowForgotPassword(true);
            }} className='text-sm text-blue-600 my-4 cursor-pointer'>Forgot password?</p>
        )}

        <button className='bg-blue-600 w-full text-white py-2 rounded-full mt-6'>
            {state === 'Login' ? 'login' : 'create account'}
        </button>

        {state === 'Login' ? (
            <p className='mt-8 text-center'>Don't have an account? <span className='text-blue-600 cursor-pointer' onClick={() => setState('Sign Up')}>Sign Up</span></p>
        ) : (
            <p className='mt-8 text-center'>Already have an account? <span className='text-blue-600 cursor-pointer' onClick={() => setState('Login')}>Login</span></p>
        )}
```

```
                <img onClick={() => setShowLogin(false)} src={assets.cross_icon}
alt="" className=' absolute top-5 right-5 cursor-pointer h-5 w-7 opacity-75 ' />
            </motion.form>
        </div>
    )
}
export default Login

import jwt from "jsonwebtoken";

const userAuth= async (req,res,next)=>{
    const{token}= req.headers;
    if(!token){
        return res.json({success:false, message: 'Not Authorized'

        })
    }
    try{
        const tokenDecode= jwt.verify(token, process.env.JWT_SECRET);
        if(tokenDecode.id){
            if (!req.body) req.body = {};
            req.body.userId = tokenDecode.id;
        }else{
            return res.json({success:false, message:'Not Authorized, Login again'})
        }

        next();
```

```
    }catch(error){
        res.json({success:false, message: error.message})
    }
}

export default userAuth
import axios from "axios"
import userModel from "../models/userModel.js"
import FormData from "form-data"
export const generateImage = async (req, res) => {
    try {
        const { userId, prompt } = req.body
        const user = await userModel.findById(userId)

        if (!user || !prompt) {
            return res.json({ success: false, message: 'Missing details' })
        }
        if (user.creditBalance == 0 || userModel.creditBalance < 0) {
            return res.json({ success: false, message: 'No Credit Balance', creditBalance: user.creditBalance })
        }
        const formData = new FormData()
        formData.append('prompt', prompt)

        const { data } = await axios.post('https://clipdrop-api.co/text-to-image/v1', formData, {
            headers: {
                'x-api-key': process.env.CLIPDROP_API,
```

```
            },
            responseType: 'arraybuffer'
        })

        const base64Image = Buffer.from(data, 'binary').toString('base64')
        const resultImage = `${base64Image}`

        await userModel.findByIdAndUpdate(user._id, {
            creditBalance
                : user.creditBalance - 1
        })

        res.json({success:true, message: "Image Generated" , creditBalance: user.creditBalance-1, resultImage})

    } catch (error) {
        console.log(error.message)
        res.json({ success: false, message: error.message })
    }
}
import bcryptjs from 'bcryptjs'; // Fixed import to match bcryptjs
import jwt from 'jsonwebtoken';
import userModel from '../models/userModel.js';
import razorpay from 'razorpay';
import transactionModel from '../models/transactionModel.js';

const registerUser = async (req, res) => {
    try {
```

```
        const { name, email, password } = req.body;

        if (!name || !email || !password) {
            return res.json({ success: false, message: 'Missing Details' });
        }

        const salt = await bcryptjs.genSalt(10); // Use bcryptjs instead of bcrypt
        const hashedPassword = await bcryptjs.hash(password, salt); // Use bcryptjs instead of bcrypt
        const userData = {
            name,
            email,
            password: hashedPassword
        };

        const newUser = userModel(userData);
        const user = await newUser.save();
        const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET);

        res.json({ success: true, token, user: { name: user.name } });

    } catch (error) {
        console.log(error);
        res.json({ success: false, message: error.message });
    }
};

const loginUser = async (req, res) => {
```

```
    try {
        const { email, password } = req.body;
        const user = await userModel.findOne({ email });

        if (!user) {
            return res.json({ success: false, message: 'User does not exist' });
        }

         const isMatch = await bcryptjs.compare(password, user.password); // Use bcryptjs here as well
        if (isMatch) {
            const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET);

            res.json({ success: true, token, user: { name: user.name } });

        } else {
            return res.json({ success: false, message: 'Invalid credentials' });

        }
    } catch (error) {
        console.log(error);
        res.json({ success: false, message: error.message });
    }
};

const userCredits = async (req, res) => {
    try {
        const { userId } = req.body;
```

```
        const user = await userModel.findById(userId);
        res.json({ success: true, credits: user.creditBalance, user: { name: user.name
} });
    } catch (error) {
        console.log(error.message);
        res.json({ success: false, message: error.message });
    }
};

const razorpayInstance = new razorpay({
    key_id: process.env.RAZORPAY_KEY_ID,
    key_secret: process.env.RAZORPAY_KEY_SECRET,
});

const paymentRazorpay = async (req, res) => {
    try {
        const { userId, planId } = req.body;

        if (!userId || !planId) {
            return res.json({
                success: false,
                message: 'Missing Details'
            });
        }

        const userData = await userModel.findById(userId);
        if (!userData) {
```

```
        return res.json({ success: false, message: 'User not found' });
    }

    let plan, credits, amount;
    switch (planId) {
        case 'Basic':
            plan = 'Basic';
            credits = 50;
            amount = 10;
            break;
        case 'Advanced':
            plan = 'Advanced';
            credits = 250;
            amount = 50;
            break;
        case 'Business':
            plan = 'Business';
            credits = 1250;
            amount = 250;
            break;
        default:
            return res.json({ success: false, message: 'Plan not found' });
    }

    const date = Date.now();

    const transactionData = {
```

```
        userId,
        plan,
        amount,
        credits,
        payment: true,
        date
    };

    const newTransaction = await transactionModel.create(transactionData);

    const options = {
        amount: amount * 100,
        currency: process.env.CURRENCY,
        receipt: newTransaction._id.toString()
    };

    razorpayInstance.orders.create(options, (error, order) => {
        if (error) {
            console.log(error);
            return res.json({
                success: false,
                message: error.message
            });
        }

        return res.json({
            success: true,
```

```
            order
        });
    });

  } catch (error) {
    console.log(error);
    res.json({
        success: false,
        message: error.message
    });
  }
};

const verifyRazorpay = async (req, res) => {
  try {
    const { razorpay_order_id } = req.body;
    const orderInfo = await razorpayInstance.orders.fetch(razorpay_order_id);

    if (orderInfo.status === 'paid') {
                                    const transactionData = await transactionModel.findById(orderInfo.receipt);
        if (transactionData.payment) { // Fixed typo: 'paymet' to 'payment'
            return res.json({ success: false, message: 'Payment Failed' });
        }
        const userData = await userModel.findById(transactionData.userId);
        const creditBalance = userData.creditBalance + transactionData.credits;
        await userModel.findByIdAndUpdate(userData._id, { creditBalance });
```

```
            await transactionModel.findByIdAndUpdate(transactionData._id, { payment: true }); // Fixed typo: 'paymet' to 'payment'
            res.json({ success: true, message: 'Credits Added' });
        } else {
            res.json({ success: false, message: 'Payment Failed' });
        }
    } catch (error) {
        console.log(error);
        res.json({ success: false, message: error.message });
    }
};

const resetPassword = async (req, res) => {
    try {
        const { email, newPassword } = req.body;

        // Find user by email
        const user = await userModel.findOne({ email });
        if (!user) {
            return res.json({ success: false, message: 'User not found' });
        }

        // Hash the new password
        const salt = await bcryptjs.genSalt(10); // Use bcryptjs here as well
        const hashedPassword = await bcryptjs.hash(newPassword, salt); // Use bcryptjs here as well

        // Update user's password
```

```
        user.password = hashedPassword;
        await user.save();

        res.json({ success: true, message: 'Password successfully changed!' });
    } catch (error) {
        console.log(error);
        res.json({ success: false, message: error.message });
    }
};

export { registerUser, loginUser, userCredits, paymentRazorpay, verifyRazorpay,
resetPassword };

import logo from './logo.svg'
import logo_icon from './logo_icon.svg'
import facebook_icon from './facebook_icon.svg'
import instagram_icon from './instagram_icon.svg'
import twitter_icon from './twitter_icon.svg'
import star_icon from './star_icon.svg'
import rating_star from './rating_star.svg'
import sample_img_1 from './sample_img_1.png'
import sample_img_2 from './sample_img_2.png'
import sample_img_3 from './sample_img_3.png'
import sample_img_4 from './sample_img_4.png'
import profile_img_1 from './profile_img_1.png'
import profile_img_2 from './profile_img_2.png'
import step_icon_1 from './step_icon_1.svg'
import step_icon_2 from './step_icon_2.svg'
```

```
import step_icon_3 from './step_icon_3.svg'
import email_icon from './email_icon.svg'
import lock_icon from './lock_icon.svg'
import cross_icon from './cross_icon.svg'
import star_group from './star_group.png'
import credit_star from './credit_star.svg'
import profile_icon from './profile_icon.png'

export const assets = {
    logo,
    logo_icon,
    facebook_icon,
    instagram_icon,
    twitter_icon,
    star_icon,
    rating_star,
    sample_img_1,
    sample_img_2,
    sample_img_3,
    sample_img_4,
    email_icon,
    lock_icon,
    cross_icon,
    star_group,
    credit_star,
    profile_icon
}
```

```
export const stepsData = [
  {
    title: 'Describe Your Vision',
    description: 'Type a phrase, sentence, or paragraph that describes the image you want to create.',
    icon: step_icon_1,
  },
  {
    title: 'Watch the Magic',
    description: 'Our AI-powered engine will transform your text into a high-quality, unique image in seconds.',
    icon: step_icon_2,
  },
  {
    title: 'Download & Share',
    description: 'Instantly download your creation or share it with the world directly from our platform.',
    icon: step_icon_3,
  },
];


export const testimonialsData = [
  {
    image:profile_img_1,
    name:'Donald Jackman',
    role:'Graphic Designer',
    stars:5,
```

```
    text:`I've been using bg.removal for nearly two years, primarily for Instagram, and it has been incredibly user-friendly, making my work much easier.`
  },
  {
    image:profile_img_2,
    name:'Richard Nelson',
    role:'Content Creator',
    stars:5,
    text:`I've been using bg.removal for nearly two years, primarily for Instagram, and it has been incredibly user-friendly, making my work much easier.`
  },
  {
    image:profile_img_1,
    name:'Donald Jackman',
    role:' Graphic Designer',
    stars:5,
    text:`I've been using bg.removal for nearly two years, primarily for Instagram, and it has been incredibly user-friendly, making my work much easier.`
  },
]


export const plans = [
  {
    id: 'Basic',
    price: 10,
    credits: 50,
    desc: 'Best for personal use.'
  },
```

```
    {
      id: 'Advanced',
      price: 50,
      credits: 250,
      desc: 'Best for business use.'
    },
    {
      id: 'Business',
      price: 250,
      credits: 1250,
      desc: 'Best for enterprise use.'
    },
  ]
{
  "name": "client",
  "private": true,
  "version": "0.0.0",
  "type": "module",
  "scripts": {
    "dev": "vite",
    "build": "vite build",
    "lint": "eslint .",
    "preview": "vite preview"
  },
  "dependencies": {
    "@heroicons/react": "^2.2.0",
    "@tailwindcss/vite": "^4.1.2",
```

```
    "axios": "^1.8.4",
    "motion": "^12.6.5",
    "react": "^19.0.0",
    "react-dom": "^19.0.0",
    "react-router-dom": "^7.4.1",
    "react-toastify": "^11.0.5"
  },
  "devDependencies": {
    "@eslint/js": "^9.21.0",
    "@types/react": "^19.0.10",
    "@types/react-dom": "^19.0.4",
    "@vitejs/plugin-react": "^4.3.4",
    "autoprefixer": "^10.4.21",
    "eslint": "^9.21.0",
    "eslint-plugin-react-hooks": "^5.1.0",
    "eslint-plugin-react-refresh": "^0.4.19",
    "globals": "^15.15.0",
    "postcss": "^8.5.3",
    "tailwindcss": "^3.4.17",
    "vite": "^6.2.0"
  }
}
```

```
import js from '@eslint/js'
import globals from 'globals'
import reactHooks from 'eslint-plugin-react-hooks'
import reactRefresh from 'eslint-plugin-react-refresh'
```

```
export default [
  { ignores: ['dist'] },
  {
    files: ['**/*.{js,jsx}'],
    languageOptions: {
      ecmaVersion: 2020,
      globals: globals.browser,
      parserOptions: {
        ecmaVersion: 'latest',
        ecmaFeatures: { jsx: true },
        sourceType: 'module',
      },
    },
    plugins: {
      'react-hooks': reactHooks,
      'react-refresh': reactRefresh,
    },
    rules: {
      ...js.configs.recommended.rules,
      ...reactHooks.configs.recommended.rules,
      'no-unused-vars': ['error', { varsIgnorePattern: '^[A-Z_]' }],
      'react-refresh/only-export-components': [
        'warn',
        { allowConstantExport: true },
      ],
    },
```

```
  },
]


@import
url('https://fonts.googleapis.com/css2?family=Acme&family=Kaushan+Script&f
amily=Merriweather:ital,wght@1,700&family=Outfit:wght@100..900&display=
swap');
@tailwind base;
@tailwind components;
@tailwind utilities;


*{
   font-family: outfit;
}
//last part


import React, { useContext } from 'react'
import { assets } from '../assets/assets'
import {motion} from 'framer-motion'
import { AppContext } from '../context/AppContext';
import { useNavigate } from 'react-router-dom';


const GenerateBtn = () => {
  const { user, setShowLogin } = useContext(AppContext);
  const navigate = useNavigate();


  const onClickHandler = () => {
    if (user) {
```

```
      navigate('/result');
    } else {
      setShowLogin(true);
    }
  }
  return (
    <motion.div className='pb-16 text-center'

    initial={{ opacity: 0.2, y: 100 }}
    transition={{ duration: 1 }}
    whileInView={{ opacity: 1, y: 0 }}
    viewport={{ once: true }}>
        <h1 className='text-2xl md:text-3xl lg:text-4xl mt-4 font-semibold text-neutral-800 py-6
      md:py-16  '>See the magic. Try now
      </h1>
      <button onClick={onClickHandler} className='inline-flex items-center gap-2 px-10 py-6 rounded-full bg-black
      text-white m-auto hover:scale-105 transition-all duration-500  text-2xl' >
        Generate Images
        <img src={assets.star_group} alt=""  className='
        h-6'/>
      </button>
    </motion.div>
  )
}

export default GenerateBtn
```

```
#!/bin/sh
basedir=$(dirname "$(echo "$0" | sed -e 's,\\,/,g')")

case `uname` in
    *CYGWIN*|*MINGW*|*MSYS*)
        if command -v cygpath > /dev/null 2>&1; then
            basedir=`cygpath -w "$basedir"`
        fi
    ;;
esac

if [ -x "$basedir/node" ]; then
  exec "$basedir/node"  "$basedir/../acorn/bin/acorn" "$@"
else
  exec node  "$basedir/../acorn/bin/acorn" "$@"
fi

import bcryptjs from 'bcryptjs'; // Fixed import to match bcryptjs
import jwt from 'jsonwebtoken';
import userModel from '../models/userModel.js';
import razorpay from 'razorpay';
import transactionModel from '../models/transactionModel.js';

const registerUser = async (req, res) => {
    try {
        const { name, email, password } = req.body;
```

```
        if (!name || !email || !password) {
            return res.json({ success: false, message: 'Missing Details' });
        }

        const salt = await bcryptjs.genSalt(10); // Use bcryptjs instead of bcrypt
        const hashedPassword = await bcryptjs.hash(password, salt); // Use bcryptjs instead of bcrypt
        const userData = {
            name,
            email,
            password: hashedPassword
        };

        const newUser = userModel(userData);
        const user = await newUser.save();
        const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET);

        res.json({ success: true, token, user: { name: user.name } });

    } catch (error) {
        console.log(error);
        res.json({ success: false, message: error.message });
    }
};

const loginUser = async (req, res) => {
    try {
```

```
        const { email, password } = req.body;
        const user = await userModel.findOne({ email });

        if (!user) {
            return res.json({ success: false, message: 'User does not exist' });
        }

        const isMatch = await bcryptjs.compare(password, user.password); // Use bcryptjs here as well
        if (isMatch) {
            const token = jwt.sign({ id: user._id }, process.env.JWT_SECRET);

            res.json({ success: true, token, user: { name: user.name } });

        } else {
            return res.json({ success: false, message: 'Invalid credentials' });

        }
    } catch (error) {
        console.log(error);
        res.json({ success: false, message: error.message });
    }
};

const userCredits = async (req, res) => {
    try {
        const { userId } = req.body;
```

```
        const user = await userModel.findById(userId);
        res.json({ success: true, credits: user.creditBalance, user: { name: user.name
} });
    } catch (error) {
        console.log(error.message);
        res.json({ success: false, message: error.message });
    }
};

const razorpayInstance = new razorpay({
    key_id: process.env.RAZORPAY_KEY_ID,
    key_secret: process.env.RAZORPAY_KEY_SECRET,
});

const paymentRazorpay = async (req, res) => {
    try {
        const { userId, planId } = req.body;

        if (!userId || !planId) {
            return res.json({
                success: false,
                message: 'Missing Details'
            });
        }

        const userData = await userModel.findById(userId);
        if (!userData) {
            return res.json({ success: false, message: 'User not found' });
```

```
    }

    let plan, credits, amount;
    switch (planId) {
        case 'Basic':
            plan = 'Basic';
            credits = 50;
            amount = 10;
            break;
        case 'Advanced':
            plan = 'Advanced';
            credits = 250;
            amount = 50;
            break;
        case 'Business':
            plan = 'Business';
            credits = 1250;
            amount = 250;
            break;
        default:
            return res.json({ success: false, message: 'Plan not found' });
    }

    const date = Date.now();

    const transactionData = {
        userId,
```

```
        plan,
        amount,
        credits,
        payment: true,
        date
    };

    const newTransaction = await transactionModel.create(transactionData);

    const options = {
        amount: amount * 100,
        currency: process.env.CURRENCY,
        receipt: newTransaction._id.toString()
    };

    razorpayInstance.orders.create(options, (error, order) => {
        if (error) {
            console.log(error);
            return res.json({
                success: false,
                message: error.message
            });
        }

        return res.json({
            success: true,
            order
```

```
            });
        });

    } catch (error) {
        console.log(error);
        res.json({
            success: false,
            message: error.message
        });
    }
};

const verifyRazorpay = async (req, res) => {
    try {
        const { razorpay_order_id } = req.body;
        const orderInfo = await razorpayInstance.orders.fetch(razorpay_order_id);

        if (orderInfo.status === 'paid') {
                                        const transactionData = await transactionModel.findById(orderInfo.receipt);
            if (transactionData.payment) { // Fixed typo: 'paymet' to 'payment'
                return res.json({ success: false, message: 'Payment Failed' });
            }
            const userData = await userModel.findById(transactionData.userId);
            const creditBalance = userData.creditBalance + transactionData.credits;
            await userModel.findByIdAndUpdate(userData._id, { creditBalance });
                await transactionModel.findByIdAndUpdate(transactionData._id, { payment: true }); // Fixed typo: 'paymet' to 'payment'
```

```
            res.json({ success: true, message: 'Credits Added' });
        } else {
            res.json({ success: false, message: 'Payment Failed' });
        }
    } catch (error) {
        console.log(error);
        res.json({ success: false, message: error.message });
    }
};

const resetPassword = async (req, res) => {
    try {
        const { email, newPassword } = req.body;

        // Find user by email
        const user = await userModel.findOne({ email });
        if (!user) {
            return res.json({ success: false, message: 'User not found' });
        }

        // Hash the new password
        const salt = await bcryptjs.genSalt(10); // Use bcryptjs here as well
          const hashedPassword = await bcryptjs.hash(newPassword, salt); // Use
bcryptjs here as well

        // Update user's password
        user.password = hashedPassword;
        await user.save();
```

```
        res.json({ success: true, message: 'Password successfully changed!' });
    } catch (error) {
        console.log(error);
        res.json({ success: false, message: error.message });
    }
};

export { registerUser, loginUser, userCredits, paymentRazorpay, verifyRazorpay,
resetPassword };
```

7. CONCLUSION

The **ImagiGen AI** application successfully fulfills its core objective of providing users with an efficient, feature-rich, and intuitive platform for generating high-quality images from text prompts using advanced AI. By leveraging modern web technologies such as the MERN stack, Clipdrop API, and Razorpay integration, the platform offers a seamless and secure experience for both creative users and developers.

Key achievements of the ImagiGen AI project include:

- A **responsive and modern user interface** that ensures compatibility across desktops, tablets, and smartphones
- **Advanced AI image generation** from textual input using Clipdrop API integration
- A **credit-based system** to manage usage and support monetization through Razorpay
- **User authentication and profile management** using secure JWT-based login systems
- An integrated **image gallery** for saving, viewing, and downloading generated images
- Support for **dark mode** and responsive design to enhance user experience
- Comprehensive **usage analytics** for tracking image generation trends and user activity

The application emphasizes usability, performance, and security, making it a powerful tool for individuals, marketers, designers, and businesses seeking to generate high-quality AI images with ease.

8. REFERENCE

- MongoDB. (2025). *MongoDB Manual*. Retrieved from https://www.mongodb.com/docs/manual
- Express.js. (2025). *Express.js Documentation*. Retrieved from https://expressjs.com
- React.js. (2025). *React Official Documentation*. Retrieved from https://react.dev
- Node.js. (2025). *Node.js Documentation*. Retrieved from https://nodejs.org/en/docs
- Clipdrop. (2025). *Clipdrop API Documentation*. Retrieved from https://clipdrop.co/apis
- Razorpay. (2025). *Razorpay Developer Docs*. Retrieved from https://razorpay.com/docs
- Tailwind CSS. (2025). *Tailwind CSS Documentation*. Retrieved from https://tailwindcss.com/docs
- Axios. (2025). *Axios GitHub Repository*. Retrieved from https://github.com/axios/axios
- OWASP Foundation. (2025). *OWASP Top 10 Web Application Security Risks*. Retrieved from https://owasp.org/www-project-top-ten/
- Nielsen Norman Group. (2023). *10 Usability Heuristics for UI Design*. Retrieved from https://www.nngroup.com/articles/ten-usability-heuristics
- GitHub. (2025). *Best Practices for Open Source Projects*. Retrieved from https://docs.github.com
- Google Developers. (2025). *Web Vitals and Performance Best Practices*. Retrieved from https://web.dev
- Banks, M. (2021). *Mastering MERN Stack: Building Modern Web Applications*. Packt Publishing.
- Banker, D. (2020). *Node.js Web Development: Server-side development with Node 14 using Express, MongoDB, and more*. Packt Publishing.
- Beighley, L., & Morrison, M. (2021). *Head First JavaScript Programming*. O'Reilly Media.
- Chan, R. (2020). *React – Up and Running: Building Web Applications*. O'Reilly Media.
- Eich, B. (2020). *JavaScript: The Definitive Guide*. O'Reilly Media.
- Hart, A. (2022). *API Development with Node.js and Express*. Apress.

- Hossain, R. (2023). *Mastering Tailwind CSS: A Utility-First CSS Framework*. Leanpub.
- Mern Stack Developers. (2022). *Full-Stack Web Development with MongoDB, Express, React, and Node*. GitHub Docs & Developer Blogs.
- Musser, R. (2021). *Implementing Payment Gateways in Web Apps: A Guide to Stripe, Razorpay, and PayPal Integration*. WebTech Press.
- O'Reilly Media. (2023). *RESTful API Design: Best Practices in API Architecture*.
- Smith, T. (2021). *Building SaaS Applications with JavaScript and Node.js*. SaaSDev Publications.
- Vercel Documentation. (2024). *Deploying and Hosting MERN Applications*. Vercel.com.
- Clipdrop Developers. (2025). *Clipdrop API Documentation*. Retrieved from https://clipdrop.co/apis
- Razorpay Developers. (2025). *Razorpay Payment Gateway Integration Guide*. Retrieved from https://razorpay.com/docs