  
  
  
1 min  
"I developed an **Employee Payroll Management System** using **Node.js, Express, and MongoDB**, focusing on backend design and scalability. The system handles **employee records, salary computation, payroll generation, and role-based authentication** for admin and employees. I structured the application in a **professional way with config, controllers, models, middleware, and routes**, and validated all endpoints using **Postman**. This project taught me how to build a production-ready backend with clean architecture, security, and real-world business logic."  
  
**3–5 Minute Detailed Talk (if asked to explain deeply)**

**🔹 1. Tech Stack & Purpose**

* **Stack:** Node.js, Express, MongoDB, JWT for authentication
* Purpose: To simulate a real-world payroll system that HR/admins can use to manage employees and salaries.

**🔹 2. Features**

* **User Authentication**: JWT-based login (Admin & Employee roles)
* **Employee Management**: CRUD operations for employees (add, update, delete, fetch)
* **Payroll System**: Salary calculation, payslip generation (stored in DB)
* **Role-Based Access**: Admin can manage all employees; employees can only view their own details
* **Security & Config**: Sensitive values stored in .env, middleware for auth validation

**🔹 3. File Structure**

* **config/** → MongoDB connection, environment setup
* **models/** → Employee, Payroll, and User schemas
* **controllers/** → Business logic (salary calculation, CRUD operations)
* **routes/** → Organized endpoints for employees, payroll, and auth
* **middleware/** → JWT auth & role-checking
* **server.js** → Entry point, connects everything

**🔹 4. Workflow Example**

1. Admin logs in → gets JWT token
2. Admin adds employee → data saved in MongoDB
3. Admin generates payroll → salary stored & accessible as payroll record
4. Employee logs in → can only view **their own payroll history**

**🔹 5. Learning Outcomes**

* Learned **professional backend structuring** (scalable file organization)
* Implemented **secure authentication & authorization**
* Worked with **Postman** for API testing and validation
* Understood how real payroll logic can be mapped to code

Text to image  
  
I built an AI-powered image generation project using Hugging Face’s Stable Diffusion. The system takes a text prompt, converts it into a representation the model can understand, and then starts from random noise, gradually refining it through multiple steps until a clear image matching the prompt is produced

For normal users, I optimized the system to generate low-quality but fast images, with extra features like multiple variations and creative prompts, so they can quickly explore ideas.

For enterprises, I designed it to produce high-quality, detailed images, with support for batch generation, making it useful for branding, marketing, or product visuals.

The project uses PyTorch with GPU acceleration, a configuration-driven design for scalability, and supports both speed-oriented and quality-oriented use cases. In short, it balances accessibility for casual users with production-grade quality for businesses."

**5 Minute Detailed Talk**

\*"This project is an AI-based image generation system built with Hugging Face’s Stable Diffusion. The main motivation was to make it usable for **two different audiences** — normal users who want quick and fun results, and enterprises that need high-quality, consistent outputs for professional use.

The **architecture** is simple but powerful. I load a Stable Diffusion pipeline into GPU memory using PyTorch. Then I built modular functions around it — generate\_image, generate\_variations, and generate\_batch.

👉 For **normal users**, the focus is on speed.

* I reduced inference steps, image size, and guidance scale, which allows the model to return results much faster.
* They can also generate multiple variations of the same prompt, which is great for creativity and experimentation.
* This is like a preview mode — users get results quickly, without waiting.

👉 For **enterprises**, the requirements are different.

* They care about **image quality and consistency**.
* I kept higher inference steps, larger image resolution, and stronger guidance scale.
* Enterprises can also submit a **batch of prompts** to generate multiple images in one go, which is useful for campaigns, e-commerce, or branding.
* The design can be extended further with fine-tuning to align with a company’s brand identity.

From a **technical perspective**,

* I used a configuration class to separate hyperparameters for user and enterprise modes.
* Random seeds ensure reproducibility, and GPU acceleration makes it efficient.
* Matplotlib is used for visualization, but this could easily be deployed into a web app using Streamlit or Gradio.

From a **real-world perspective**,

* This system helps normal users by making AI art accessible, while enterprises can integrate it into content pipelines.
* It demonstrates how to balance **performance and quality trade-offs** depending on the end user.

In short, the project is not just a model demo but a **practical design**, showing how AI systems can be adapted for different stakeholders — casual users and businesses.

Code:  
!pip install --upgrade diffusers transformers -q

from pathlib import Path

import torch

from diffusers import StableDiffusionPipeline

import matplotlib.pyplot as plt

# ---------------- CONFIG ---------------- #

class CFG:

device = "cuda"

seed = 42

generator = torch.Generator(device).manual\_seed(seed)

image\_gen\_model\_id = "stabilityai/stable-diffusion-2"

image\_gen\_guidance\_scale = 9

user\_steps = 15 # low quality (fast)

user\_size = (256, 256)

user\_guidance = 6

enterprise\_steps = 40 # high quality (detailed)

enterprise\_size = (512, 512)

enterprise\_guidance = 9

# ---------------- MODEL LOADING ---------------- #

image\_gen\_model = StableDiffusionPipeline.from\_pretrained(

CFG.image\_gen\_model\_id,

torch\_dtype=torch.float16,

revision="fp16",

use\_auth\_token='hf\_LXqwEIUlnDFuwUadrnznFuReyjZFNFYYKz'

)

image\_gen\_model = image\_gen\_model.to(CFG.device)

# ---------------- FUNCTIONS ---------------- #

def generate\_image(prompt, model, mode="user"):

"""

mode = "user" -> Low quality, fast

mode = "enterprise" -> High quality, detailed

"""

if mode == "user":

steps = CFG.user\_steps

size = CFG.user\_size

guidance = CFG.user\_guidance

else:

steps = CFG.enterprise\_steps

size = CFG.enterprise\_size

guidance = CFG.enterprise\_guidance

image = model(

prompt,

num\_inference\_steps=steps,

generator=CFG.generator,

guidance\_scale=guidance

).images[0]

image = image.resize(size)

return image

def generate\_variations(prompt, model, n=3):

"""Generate multiple variations (useful for normal users)"""

images = []

for i in range(n):

image = model(

prompt,

num\_inference\_steps=CFG.user\_steps,

generator=torch.Generator(CFG.device).manual\_seed(CFG.seed + i),

guidance\_scale=CFG.user\_guidance

).images[0]

image = image.resize(CFG.user\_size)

images.append(image)

return images

def generate\_batch(prompts, model):

"""Generate batch images (useful for enterprises)"""

results = {}

for p in prompts:

results[p] = generate\_image(p, model, mode="enterprise")

return results

# ---------------- DEMO ---------------- #

# Normal user - quick preview

img1 = generate\_image("a man with a melon", image\_gen\_model, mode="user")

plt.imshow(img1)

plt.title("User Mode (Low Quality)")

plt.axis("off")

plt.show()

# Enterprise - high quality

img2 = generate\_image("a man with a melon", image\_gen\_model, mode="enterprise")

plt.imshow(img2)

plt.title("Enterprise Mode (High Quality)")

plt.axis("off")

plt.show()

# Variations for normal user

var\_images = generate\_variations("a futuristic city at night", image\_gen\_model, n=3)

for i, v in enumerate(var\_images):

plt.imshow(v)

plt.title(f"Variation {i+1}")

plt.axis("off")

plt.show()

# Batch generation for enterprise

batch\_prompts = ["a cat wearing sunglasses", "a car in cyberpunk style"]

batch\_results = generate\_batch(batch\_prompts, image\_gen\_model)

for p, img in batch\_results.items():

plt.imshow(img)

plt.title(f"Enterprise Batch: {p}")

plt.axis("off")

plt.show()