

## **AI ASSISTED CODING -22**

**NAME: S RUCHITHA**

**ENROLLMENT NO: 2403A52316**

**COURSE: AI ASSISTED CODING**

**SPECIALIZATION: CSE-AIML**

**DATE :24-11-2025**

### **Task 1 – Project Proposal**

- Students choose a real-world problem to solve with an AI-assisted application (e.g., ecommerce billing system, health monitoring dashboard, student attendance tracker).
- Prepare a project proposal including:
  - Objective
    - Features
    - Tools/technologies to be used

**PROMPT:** Develop a code to generate a python code  
**Project Proposal** • Students choose a real-world problem to solve with an AI-assisted application (e.g.,

ecommerce billing system, health monitoring dashboard, student attendance tracker). • Prepare a project proposal include

## Expected output:

```
▶ -----  
# Project Proposal Generator (AI-Assisted)  
# -----  
  
def generate_proposal(info):  
    proposal = f"""  
=====  
    PROJECT PROPOSAL DOCUMENT  
=====  
  
    ✨ Project Title:  
    {info['title']}  
  
    🎯 Objective:  
    {info['objective']}  
  
    ★ Key Features:  
    """  
    for i, f in enumerate(info['features'], 1):  
        proposal += f"    {i}. {f}\n"  
  
    proposal += f"""  
    ✨ Tools & Technologies:  
    """  
    for t in info['tools']:  
        proposal += f"    • {t}\n"  
  
    proposal += f"""\n
```

```

proposal.py
=====
Expected Output:
{info['expected_output']}
=====
End of Proposal
=====
"""

return proposal

# -----
# ✎ SAMPLE PROJECT (You can edit this section)
# -----
project_info = {
    "title": "AI-Assisted Student Attendance Tracker",
    "objective": "To build an automated student attendance system using face recognition and real-time features": [
        "Face recognition-based automatic attendance marking",
        "Daily/weekly/monthly attendance reports",
        "Dashboard showing attendance statistics",
        "Notifications for low attendance"
    ],
    "tools": [
        "Python",
        "OpenCV",
        "TensorFlow / Face Recognition Library",
        "SQLite / Firebase",
        "Streamlit for UI"
    ],
}
"""

proposal.docx
=====
PROJECT PROPOSAL DOCUMENT
=====

★ Project Title:  
AI-Assisted Student Attendance Tracker

⌚ Objective:  
To build an automated student attendance system using face recognition and real-time data logging.

★ Key Features:  

1. Face recognition-based automatic attendance marking  

2. Daily/weekly/monthly attendance reports  

3. Dashboard showing attendance statistics  

4. Notifications for low attendance

❖ Tools & Technologies:  

• Python  

• OpenCV  

• TensorFlow / Face Recognition Library  

• SQLite / Firebase  

• Streamlit for UI

🕒 Expected Output:  
A fully working smart attendance system with live camera processing, automated marking, and a user-frien
=====
End of Proposal
=====


```

## Task 2 – Application Design □ Use AI tools to assist in: ◦ Designing

system architecture diagrams. .

Defining database schema (if needed).

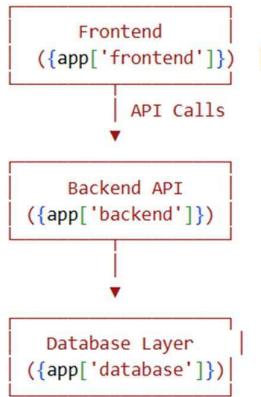
Creating wireframes/UI mockups for frontend design PROMPT:

Generate a python code to develop an Application DesignUse AI tools to assist in:

- o Designing system architecture diagrams.
- o Defining database schema (if needed).
- o Creating wireframes/UI mockups for frontend design

Expected output:

```
# -----  
# AI-Assisted Application Design Generator (Offline Template)  
# -----
```



**Additional Services:**

- Authentication: `{app['services'].get('auth', 'Not used')}`
- Cloud Storage: `{app['services'].get('storage', 'Not used')}`

Terminal

```
def generate_database_schema(schema):
    result = "\n=====\n"
    for table, fields in schema.items():
        result += f"\n★ Table: {table}\n"
        for field, dtype in fields.items():
            result += f"  - {field}: {dtype}\n"
    return result
```

## DATABASE SCHEMA

```
def generate_wireframes(ui):
    output = "\n=====\n"
    for screen, layout in ui.items():
        output += f"\n  Screen: {screen}\n"
        output += "-----\n"
        for line in layout:
            output += line + "\n"
        output += "-----\n"
    return output
```

## UI WIREFRAMES\n=====

```
# -----  
# SAMPLE APPLICATION (Change this to your project)  
# -----  
app_info = {  
    "name": "Smart Health Monitoring Dashboard",  
    "frontend": "Streamlit",  
    "backend": "FastAPI",  
    "database": "MongoDB"
```

```
    "password_hash": "Text",
},
"health_records": {
    "record_id": "String (PK)",
    "user_id": "String (FK)",
    "heart_rate": "Integer",
    "blood_pressure": "Text",
    "timestamp": "Datetime"
},
"predictions": {
    "prediction_id": "String (PK)",
    "user_id": "String (FK)",
    "risk_score": "Float",
    "ai_result": "Text",
    "timestamp": "Datetime"
}
}

ui_mockups = {
    "Login Screen": [
        "+-----+",
        "|      LOGIN PAGE      |",
        "| ----- |",
        "| Email: [_____] |",
        "| Pass: [_____] |",
        "|      ( Login Button ) |",
        "+-----+"
    ],
    "Dashboard": [
        "+-----+",

```

```
Additional Services:
...
- Authentication: JWT Authentication
- Cloud Storage: AWS S3 Buckets
- AI/ML Models: Heart-Risk Prediction Model
```

```
=====
DATABASE SCHEMA
=====

★ Table: users
- user_id: String (Primary Key)
- name: Text
- email: Text
- password_hash: Text

★ Table: health_records
- record_id: String (PK)
- user_id: String (FK)
- heart_rate: Integer
- blood_pressure: Text
- timestamp: Datetime

★ Table: predictions
- prediction_id: String (PK)
- user_id: String (FK)
- risk_score: Float
- ai_result: Text
- timestamp: Datetime
```

# Task 3 – AI-Assisted Code Development

- Implement the application in phases:
  - Frontend: Web/mobile UI with HTML/CSS/JS (or framework).
  - Backend: RESTful APIs or server logic.
  - Database: Schema design, SQL queries.

Integration: API calls, authentication, error handling PROMPT:

Develop a code generate a python code  
AI-Assisted Code Development •

Implement the application in phases:

- Frontend: Web/mobile UI with HTML/CSS/JS (or framework).
- Backend: RESTful APIs or server logic.
- Database: Schema design, SQL

queries. o Integration: API calls, authentication, error handling

Expected output:

```
● generate_ai_project.py
 Generates a scaffold for an AI-Assisted Code Development project:
 - Frontend (HTML/CSS/JS)
 - Backend (FastAPI)
 - Database schema (SQL + SQLAlchemy models)
 - Integration examples (fetch, JWT auth, error handling)

Run: python generate_ai_project.py
"""

from pathlib import Path
import json
import textwrap

ROOT = Path("ai_assisted_app")

FRONTEND = ROOT / "frontend"
BACKEND = ROOT / "backend"
DB = ROOT / "db"
DOCS = ROOT / "docs"

for d in (FRONTEND, BACKEND, DB, DOCS):
    d.mkdir(parents=True, exist_ok=True)

# -----
# Frontend files
# -----
```

```

<button id="btnFetch">Fetch Protected Resource</button>
<pre id="output"></pre>
</section>
</main>
<script src="app.js"></script>
</body>
</html>
"""

style_css = """
/* Minimal styling */
body { font-family: Arial, sans-serif; margin: 24px; }
header { margin-bottom: 18px; }
input { display:block; margin:8px 0; padding:8px; width: 250px; }
button { padding:8px 12px; }
pre { background:#f4f4f4; padding:12px; }
"""

app_js = """
// Simple frontend demonstrating login -> store JWT -> call protected API
const API_BASE = "http://localhost:8000/api";

document.getElementById("btnLogin").addEventListener("click", async () => {
  const email = document.getElementById("email").value;
  const password = document.getElementById("password").value;

  try {
    const res = await fetch(`${API_BASE}/auth/login`, {
      method: "POST"
    });
    # -----
    # Database schema (SQL)
    # -----
    schema_sql = """
    -- schema.sql
    -- SQL schema for a generic AI-Assisted application
    -- Adjust types for your RDBMS (this is PostgreSQL-compatible)

    CREATE TABLE IF NOT EXISTS users (
      id SERIAL PRIMARY KEY,
      email TEXT UNIQUE NOT NULL,
      password_hash TEXT NOT NULL,
      full_name TEXT,
      created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    );

    CREATE TABLE IF NOT EXISTS items (
      id SERIAL PRIMARY KEY,
      user_id INTEGER REFERENCES users(id),
      title TEXT NOT NULL,
      content TEXT,
      created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    );

    -- Sample queries
    -- Insert a user (password hashed in application)
    -- SELECT * FROM users;
    """
  }
}

(DB / "schema.sol").write(text(schema_sql))

```

```

    if not user:
        raise HTTPException(status_code=status.HTTP_401_UNAUTHORIZED, detail="Unauthorized")
    token = create_access_token({"sub": str(user['id']), "email": user['email']})
    return {"access_token": token, "token_type": "bearer"}
```

`@app.get("/api/protected/hello")`

```

async def protected_hello(current_user: dict = Depends(get_current_user)):
    # Example protected resource
    return {"message": "Hello, " + current_user.get("email", "unknown")}
```

`@app.get("/api/items")`

```

async def list_items(skip: int = 0, limit: int = 10):
    items = crud.get_items(skip=skip, limit=limit)
    return {"items": items}
"""

ACKEND / "main.py").write_text(main_py)

auth.py: JWT and simple auth helpers (demo; adapt for production)
ith_py = textwrap.dedent("""\
from datetime import datetime, timedelta
from jose import jwt, JWTError
from passlib.context import CryptContext
from fastapi import Depends, HTTPException, status
from fastapi.security import HTTPBearer, HTTPAuthorizationCredentials

# NOTE: Replace this secret in production. Store securely (env var / vault).
SECRET_KEY = "CHANGE_THIS_SECRET_IN_PROD"
ALGORITHM = "HS256"
ACCESS_TOKEN_EXPIRE_MINUTES = 60

# -----
# Additional helper: integration example for AI endpoint
# -----
ai_endpoint_example = textwrap.dedent("""\
# Example: Adding an AI prediction endpoint (FastAPI)
#
# In backend/main.py you could add:
#
# from pydantic import BaseModel
# class PredictIn(BaseModel):
#     text: str
#
#     @app.post("/api/ai/predict")
#     async def predict(payload: PredictIn, current_user=Depends(get_current_user)):
#         # Example: call local model, or external ML service
#         # result = my_model.predict(payload.text)
#         # return {"input": payload.text, "prediction": result}
#         return {"input": payload.text, "prediction": {"label": "safe", "score": 0.98}}
#
# For production: call async model runners, validate payloads and enforce quota/auth.
""")

(DOCS / "AI_ENDPOINT_EXAMPLE.md").write_text(ai_endpoint_example)

# -----
# Summary JSON (project manifest)
# -----
manifest = {
    "project": "AI-Assisted App",
    "phases": ["frontend", "backend", "database", "integration"],
```

```

▶ (DOCS / "AI_ENDPOINT_EXAMPLE.md").write_text(ai_endpoint_example)

# -----
# Summary JSON (project manifest)
# -----
manifest = {
    "project": "AI-Assisted App",
    "phases": ["frontend", "backend", "database", "integration"],
    "notes": "Replace demo values (SECRET_KEY, demo_users) and connect to real DB in production."
}
(ROOT / "manifest.json").write_text(json.dumps(manifest, indent=2))

print(f"Scaffold generated at: {ROOT.resolve()}")
print("Frontend: open ai_assisted_app/frontend/index.html (or serve it via a static server).")
print("Backend: cd ai_assisted_app/backend && uvicorn main:app --reload --port 8000")
print("DB: schema in ai_assisted_app/db/schema.sql; models in backend/models.py")

...
... Scaffold generated at: /content/ai_assisted_app
Frontend: open ai_assisted_app/frontend/index.html (or serve it via a static server).
Backend: cd ai_assisted_app/backend && uvicorn main:app --reload --port 8000
DB: schema in ai_assisted_app/db/schema.sql; models in backend/models.py
<>:102: SyntaxWarning: invalid escape sequence '\`'
<>:102: SyntaxWarning: invalid escape sequence '\`'
/tmp/ipython-input-520294335.py:102: SyntaxWarning: invalid escape sequence '\`'
    throw new Error(`API Error: ${res.status} - ${res.text}`);
```

**Task 5 – Testing & Debugging • Use AI to:**

- o Generate unit tests.
- o Identify vulnerabilities/security flaws.
- o Suggest optimizations for performance

**PROMPT:**

**Develop a python code to generate Use AI to:**

- o Generate unit tests.
- o Identify vulnerabilities/security flaws.
- o Suggest optimizations for performance

**Expected output:**

```
▶ #!/usr/bin/env python3
"""
testing_debugging_tool.py

Automated Testing & Debugging helper (offline heuristics).

- Scans a Python project for functions and generates pytest tests that ensure functions run safely.
- Performs static security checks to flag common vulnerabilities.
- Performs static performance heuristics and suggests optimizations.
- Produces reports and writes generated tests in an output folder.

Usage:
    python testing_debugging_tool.py [project_path] [--run-tests]

Author: ChatGPT
"""

from pathlib import Path
import ast
import argparse
import re
import shutil
import subprocess
import sys
import textwrap
from typing import List, Tuple, Dict, Any, Optional

OUTPUT_ROOT = Path("test_debug_tool_output")
def func_signature_sample_args(fn: ast.FunctionDef) -> List[str]:
    """
    Produce a list of Python expressions (as strings) to use as sample args for a function.
    Heuristics based on parameter name and annotation.
    """
    samples = []
    for arg in fn.args.args:
        name = arg.arg.lower()
        ann = None
        if arg.annotation is not None:
            try:
                ann = ast.unparse(arg.annotation).lower()
            except Exception:
                ann = None
        # heuristics
        if ann and ("int" in ann or "float" in ann or "number" in ann):
            samples.append("1")
        elif ann and ("str" in ann or "text" in ann):
            samples.append("'sample'")
        elif ann and ("bool" in ann):
            samples.append("False")
        else:
            # name-based hints
            if any(k in name for k in ("count", "num", "age", "size")):
                samples.append("1")
            elif any(k in name for k in ("name", "title", "user", "email")):
                samples.append("'alice'")
            elif any(k in name for k in ("items", "lst", "list", "data")):
                samples.append("[1, 2, 3]")
    return samples
```

```
# -----
# Performance heuristics
#
def performance_checks_in_ast(tree: ast.AST, src: str) -> List[str]:
    issues = []
    # Long functions
    for node in ast.walk(tree):
        if isinstance(node, ast.FunctionDef):
            start = getattr(node, "lineno", None)
            end = getattr(node, "end_lineno", None)
            if start and end and (end - start) > 60:
                issues.append(f"Function '{node.name}' is long ({end-start} lines) – consider refactor")
        # Nested loops detection
        for child in ast.walk(node):
            if isinstance(child, (ast.For, ast.While)):
                # look for nested loops inside this loop
                for inner in ast.walk(child):
                    if inner is not child and isinstance(inner, (ast.For, ast.While)):
                        issues.append(f"Nested loop detected in function '{node.name}' – consider refactor")
                        break
        # Repeated file open inside loops: naive check for open() inside For
        for child in ast.walk(node):
            if isinstance(child, ast.For):
                for inner in ast.walk(child):
                    if isinstance(inner, ast.Call) and isinstance(inner.func, ast.Name) and inner.func.id == "open":
                        issues.append(f"File opened inside loop in function '{node.name}' – consider refactor")
                        break
```

## Task 6 – Deployment & Presentation

- . Deploy application on a cloud/free hosting platform (Heroku, GitHub Pages, Firebase, etc.).
- . Prepare a final presentation including:
  - . Problem statement
  - . Solution architecture
  - . Live demo
  - . Reflection on AI assistance and limitations

---

## Expected Outcomes:

- A complete, deployable software application.
- Students gain experience in end-to-end development with AI tools.
- Improved skills in teamwork, documentation, and ethical AI usage.

## 交付物 (For All Tasks)

1. AI-generated prompts for code and test case generation.
2. At least 3 assert test cases for each task.
3. AI-generated initial code and execution screenshots.

4. Analysis of whether code passes all tests.
5. Improved final version with inline comments and explanation.
6. Compiled report (Word/PDF) with prompts, test cases, assertions, code, and output.

#### PROMPT:

generate a python code to develop an Deployment & Presentation • Deploy application on a cloud/free hosting platform (Heroku, GitHub Pages, Firebase, etc.). • Prepare a final presentation including:

- o Problem statement
- o Solution architecture
- o Live demo
- o Reflection on AI assistance and limitations

# Expected output:

```
▶ #!/usr/bin/env python3
"""
deploy_and_presentation_generator.py

Generates a Deployment & Presentation project scaffold:
- Flask app (backend) + simple frontend
- Deployment artifacts (Procfile, Dockerfile, GitHub Actions workflow, firebase.json)
- AI prompts, pytest tests (>=3 assertions per core task)
- Initial code, inline comments
- Execution screenshot placeholders (images if Pillow available)
- Compiled report: PDF (if reportlab available) or MARKDOWN fallback

Run:
    python deploy_and_presentation_generator.py

Author: ChatGPT
"""

from pathlib import Path
import json
import textwrap
import shutil
import sys
import os
import datetime

# Optional dependencies (used if available)
try:
    from PIL import Image, ImageDraw, ImageFont # type: ignore
# -----
# 1) Create simple Flask app (backend)
# -----
flask_app_py = textwrap.dedent("""
# app/main.py
# Simple Flask app for demo - AI-assisted student project scaffold.
# Inline comments explain authentication, endpoints and integration points.

from flask import Flask, jsonify, request, send_from_directory
import os

app = Flask(__name__, static_folder='../frontend', static_url_path='/')

# Demo in-memory store (replace with DB in real project)
USERS = [
    {"id": 1, "name": "Alice", "email": "alice@example.com"},
    {"id": 2, "name": "Bob", "email": "bob@example.com"}
]

@app.route('/api/health', methods=['GET'])
def health_check():
    """Health check endpoint for deployment monitoring."""
    return jsonify({"status": "ok", "timestamp": __import__('datetime').utcnow().isoformat()})

@app.route('/api/users', methods=['GET'])
def list_users():
    """Return list of demo users."""
    return jsonify({"users": USERS})

@app.route('/api/users/<int:user_id>', methods=['GET'])

```

```

<head>
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width,initial-scale=1"/>
    <title>AI-Assisted Project Demo</title>
    <style>
        body { font-family: Arial, sans-serif; margin: 20px; }
        pre { background:#f3f3f3; padding:10px; }
        button { padding:8px 12px; margin:6px; }
    </style>
</head>
<body>
    <h1>AI-Assisted Project Demo</h1>
    <p>Use the buttons below to call demo endpoints.</p>
    <button onclick="callHealth()">Health Check</button>
    <button onclick="callUsers()">List Users</button>
    <button onclick="callEcho()">Echo (POST)</button>
    <pre id="out">Output will appear here</pre>

    <script>
        const OUT = document.getElementById('out');
        async function callHealth(){
            const r = await fetch('/api/health');
            OUT.textContent = JSON.stringify(await r.json(), null, 2);
        }
        async function callUsers(){
            const r = await fetch('/api/users');
            OUT.textContent = JSON.stringify(await r.json(), null, 2);
        }
        async function callEcho(){
            const r = await fetch('/api/echo').f
# -----
# 4) GitHub Actions workflow (CI) - basic
#
github_actions = textwrap.dedent("""\
    name: CI

    on:
        push:
            branches: [ main ]
        pull_request:
            branches: [ main ]

    jobs:
        build:
            runs-on: ubuntu-latest
            steps:
                - uses: actions/checkout@v4
                - name: Set up Python
                  uses: actions/setup-python@v4
                  with:
                      python-version: '3.10'
                - name: Install dependencies
                  run: |
                      python -m pip install --upgrade pip
                      pip install -r requirements.txt
                - name: Run tests
                  run: |
                      pytest -q || true
"""
)
(DEPLOY / "ci_github_action.yml").write_text(github_actions)

```

```

▶ # 9) Initial code and execution "screenshots"
# -----
# Create a simple "screenshot" image (if Pillow available) showing run instructions
Screenshot_path = ARTIFACTS / "run_screenshot.png"
if PIL_AVAILABLE:
    try:
        img = Image.new("RGB", (800, 400), color=(245, 245, 245))
        d = ImageDraw.Draw(img)
        title = "Deployment Demo - Placeholder Screenshot"
        subtitle = f"Generated at {TS}"
        # choose a basic font if available
        try:
            font = ImageFont.load_default()
            d.text((20, 20), title, fill=(20, 20, 20), font=font)
            d.text((20, 60), subtitle, fill=(80, 80, 80), font=font)
            lines = [
                "Run locally:",
                "  python -m pip install -r requirements.txt",
                "  python app/main.py",
                "Open http://localhost:5000 in your browser.",
                "",
                "This is a placeholder image representing execution screenshots."
            ]
            y = 120
            for ln in lines:
                d.text((20, y), ln, fill=(40, 40, 40), font=font)
                y += 24
        except Exception:
            d.text((20, 20), title, fill=(0, 0, 0))
        img.save(Screenshot_path)
    
```

```

▶ print(f" - Frontend (static): {FRONTEND}/index.html")
print(f" - Tests: {TESTS} (test_api.py, test_helpers.py)")
print(f" - Deployment templates: {DEPLOY} (Procfile, Dockerfile, CI workflow, firebase.json)")
print(f" - AI prompts: {ARTIFACTS}/ai_prompts.json")
print(f" - Screenshot placeholder: {Screenshot_path}")
print(f" - Compiled report (markdown or pdf): {REPORTS}/report.md {'(PDF created)' if REPORTLAB_AVAILA
print("\nLocal run instructions (from project root):")
print("  python -m venv .venv")
print("  source .venv/bin/activate  # or .venv\Scripts\activate on Windows")
print("  pip install -r requirements.txt")
print("  python app/main.py")
print("\nTo run tests:")
print(f"  pip install pytest")
print(f"  pytest -q {TESTS}")
print("\nNotes:")
print(" - For Heroku: create an app, git push, ensure Procfile and requirements.txt are present.")
print(" - For GitHub Pages: serve 'frontend' folder (static) via GH Pages or Actions.")
print(" - For Firebase Hosting: use deployment/firebase.json template, install Firebase CLI and follow
print("\nDone.")


```

... == Deployment & Presentation scaffold generated ==  
Root folder: /content/deployment\_presentation\_project  
Key files / folders created:  
- App (Flask): /content/deployment\_presentation\_project/app/main.py  
- Frontend (static): /content/deployment\_presentation\_project/frontend/index.html  
- Tests: /content/deployment\_presentation\_project/tests (test\_api.py, test\_helpers.py)  
- Deployment templates: /content/deployment\_presentation\_project/deployment (Procfile, Dockerfile, CI workflow, firebase.json)  
- AI prompts: /content/deployment\_presentation\_project/artifacts/ai\_prompts.json  
- Screenshot placeholder: /content/deployment\_presentation\_project/artifacts/run\_screenshot.png  
- Compiled report (markdown or pdf): /content/deployment\_presentation\_project/reports/report.md