Requirement Analysis

1. Objective

The requirement analysis phase aims to gather, analyze, and document the functional and non-functional requirements of the system. It ensures that the solution aligns with stakeholder expectations and provides a clear foundation for design and implementation.

2. Key Activities

Stakeholder Interviews & Workshops

Collect requirements from students, teachers, administrators, and other stakeholders.

Requirement Gathering

Identify and document user needs, system features, and constraints.

Use Case Development

Define scenarios describing how users will interact with the system.

Requirement Classification

Functional Requirements: Features and system behavior.

Non-Functional Requirements: Performance, security, scalability, usability, etc.

Requirement Validation

Review requirements with stakeholders to confirm accuracy and feasibility.

3. Deliverables

Software Requirement Specification (SRS) – Comprehensive document detailing requirements.

Use Case Diagrams & Descriptions – Visual and textual representation of user-system interactions.

User Stories / Epics – Agile-friendly requirement breakdowns.

Acceptance Criteria – Conditions that must be met for a requirement to be considered complete.

Requirement Traceability Matrix (RTM) – Maps requirements to design, implementation, and testing phases.

4. Tools & Techniques

Requirement Management: Jira, Confluence, IBM DOORS.

Diagramming Tools: Lucidchart, Draw.io, Microsoft Visio.

Collaboration: Google Docs, Notion, Miro.

5. Expected Outcomes

A validated and documented list of system requirements.

Reduced ambiguity and miscommunication between stakeholders and developers.

Clear traceability from requirements to design and testing.

A baseline document (SRS) to guide the Project Design Phase.

PROJECT PLANNING

1. Objective

The objective of the project planning phase is to define the scope, milestones, timelines, and resources required to execute the project successfully. It ensures that all stakeholders are aligned, risks are considered, and deliverables are clearly scheduled.

2. Key Activities

Define Scope & Goals

Establish project objectives, scope boundaries, and expected outcomes.

Work Breakdown Structure (WBS)

Break down the project into tasks, sub-tasks, and milestones.

Resource Allocation

Assign roles, responsibilities, and tools needed for development, testing, and deployment.

Scheduling & Timeline

Develop a timeline using Gantt charts or agile sprint planning.

Risk Management

Identify potential risks (technical, financial, timeline delays) and prepare mitigation strategies.

Communication Plan

Define how progress will be reported (meetings, dashboards, weekly updates).

3. Deliverables

Project Plan Document – Consolidates scope, timelines, resources, and risks.

Gantt Chart / Roadmap – Visual representation of milestones and deadlines.

Resource Plan – Roles, responsibilities, and team allocation.

Risk Assessment Matrix – Identified risks with severity, likelihood, and mitigation.

Budget & Cost Estimates – Projected costs of tools, cloud infrastructure, and resources.

4. Tools & Technologies

Project Management Tools: Jira, Trello, Asana, Microsoft Project.

Timeline & Roadmap Tools: GanttProject, Monday.com, ClickUp

Collaboration Tools: Slack, Microsoft Teams, Google Workspace.

5. Expected Outcomes

A clear roadmap for development and testing.

Realistic timelines with milestones.

Defined responsibilities for each team member.

Risk mitigation strategies to avoid project delays.

Stakeholder confidence in project feasibility.

PROJECT DESIGN PHASE

1. Objective

The purpose of the design phase is to translate requirements into a detailed technical blueprint that guides development. It ensures the system's architecture, components, and interfaces are well-structured, scalable, and aligned with project goals.

2. Key Activities

High-Level Design (HLD)

Define overall system architecture (frontend, backend, database, APIs).

Identify major modules and their responsibilities.

Establish technology stack.

Low-Level Design (LLD)

Detailed module-level design, class diagrams, and data models.

Define workflows, algorithms, and logic for each component.

API contracts and data flow diagrams.

Database Design

Entity-Relationship Diagram (ERD).

Schema definition and normalization.

Indexing and optimization plans.

UI/UX Design

Wireframes and mockups for key screens.

Navigation flow diagrams.

Accessibility and usability considerations.

Security & Compliance Design

Define authentication, authorization, and data encryption mechanisms.

Ensure compliance with relevant regulations (GDPR, HIPAA, etc., if applicable).

3. Deliverables

System Architecture Diagram – Visual representation of major components and interactions.

Database ER Diagram – Defines entities, relationships, and schema.

Module Specifications – Documentation of responsibilities and interactions of each module.

UI/UX Prototypes – Wireframes/mockups for user interface.

Design Document – Consolidated document covering HLD, LLD, database, and UI/UX.

4. Tools & Technologies

Design Tools: Figma, Adobe XD, Balsamiq, Lucidchart.

Architecture/Diagram Tools: Draw.io, Microsoft Visio.

Database Design: MySQL Workbench, dbdiagram.io, ERDPlus.

5. Expected Outcomes

Clear technical roadmap for developers.

Reduced ambiguity during coding.

A design that ensures scalability, maintainability, and security.

Stakeholder alignment on how the final system will look and function.

PERFORMANCE TESTING

1. Objective

The objective of performance testing is to evaluate how the system behaves under various levels of load and stress. It ensures that the application is stable, scalable, and capable of delivering a smooth user experience without performance degradation.

2. Key Activities

Test Planning

Define performance criteria, test scenarios, and success benchmarks.

Environment Setup

Configure hardware, software, and network conditions to simulate real usage.

Load Testing

Simulate multiple users accessing the system simultaneously to measure response times and throughput.

Stress Testing

Push the system beyond its normal operating capacity to identify breaking points.

Scalability Testing

Assess whether the system can handle increased load when more resources (servers, CPUs, memory) are added.

Endurance (Soak) Testing

Run the system for extended periods to detect memory leaks, performance degradation, or stability issues.

3. Tools & Technologies

Common tools for performance testing:

Apache JMeter – for simulating concurrent users.

Locust – for Python-based load testing.

k6 – lightweight load testing tool.

Monitoring Tools – Grafana, Prometheus, New Relic, or CloudWatch to track resource usage.

4. Metrics to Measure

Response Time – Time taken to process a user request.

Throughput – Number of transactions handled per second.

Error Rate – Percentage of failed requests.

CPU & Memory Utilization – Resource consumption under load.

Scalability – System's ability to handle increased traffic by scaling resources.

5. Deliverables

Performance Test Plan – Outlines test objectives, scope, tools, and strategy.

Test Scripts & Scenarios – Automated scripts for simulating user activity.

Performance Test Report – Includes results, bottleneck analysis, and recommendations.

Optimization Roadmap – Suggested improvements for better performance.

6. Expected Outcomes

Identification of performance bottlenecks (slow queries, heavy processes, memory leaks).

Verified system capability to handle target user loads.

Improved confidence in system stability before deployment.

IDEATION PHASE

1. Objective

The ideation phase aims to clearly define the problem, explore possible solutions, and establish the vision for the project. At this stage, brainstorming and initial research are performed to ensure the project addresses a real need and is aligned with user requirements and sustainability goals.

2. Key Activities

Problem Identification

Understand the challenges faced by the target audience (e.g., students, educators, policymakers).

Brainstorming Solutions

Generate multiple potential approaches to solving the identified problem.

Competitor Analysis

Study similar tools and platforms to identify gaps and opportunities.

Feasibility Study

Assess the technical, financial, and resource feasibility of proposed ideas.

Define Vision and Scope

Create a clear, high-level vision of the project's purpose and limitations.

3. Deliverables

Problem Statement – A precise description of the problem being solved.

Vision Document – Defines what success looks like for the project.

Feature Wishlist – A list of potential features that could be implemented.

Preliminary User Profiles – Early assumptions about end users and their needs.

High-Level Use Cases – Scenarios describing how users would interact with the system.

4. Expected Outcomes

Alignment among stakeholders about the project's purpose and direction.

A prioritized list of potential features for later refinement.

A foundation for moving into the Requirement Analysis Phase.

CODING:

```
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
import PyPDF2
import io

# Load model and tokenizer
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto" if torch.cuda.is_available() else None
)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)

    if torch.cuda.is_available():
        inputs = {k: v.to(model.device) for k, v in inputs.items()}

    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_length=max_length,
```

```
            temperature=0.7,
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id
        )

    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    response = response.replace(prompt, "").strip()
    return response

def extract_text_from_pdf(pdf_file):
    if pdf_file is None:
        return ""

    try:
        pdf_reader = PyPDF2.PdfReader(pdf_file)
        text = ""
        for page in pdf_reader.pages:
            text += page.extract_text() + "\n"
        return text
    except Exception as e:
        return f"Error reading PDF: {str(e)}"

def eco_tips_generator(problem_keywords):
    prompt = f"Generate practical and actionable eco-friendly tips for sustainable living related to: {problem_keywords}. Provide specific solutions and suggestions:"
    return generate_response(prompt, max_length=1000)

def policy_summarization(pdf_file, policy_text):
    # Get text from PDF or direct input
```

```
    if pdf_file is not None:
        content = extract_text_from_pdf(pdf_file)
        summary_prompt = f"Summarize the following policy document and extract the most important points, key provisions, and implications:\n\n{content}"
    else:
        summary_prompt = f"Summarize the following policy document and extract the most important points, key provisions, and implications:\n\n{policy_text}"

    return generate_response(summary_prompt, max_length=1200)

# Create Gradio interface
with gr.Blocks() as app:
    gr.Markdown("# Eco Assistant & Policy Analyzer")

    with gr.Tabs():
        with gr.TabItem("Eco Tips Generator"):
            with gr.Row():
                with gr.Column():
                    keywords_input = gr.Textbox(
                        label="Environmental Problem/Keywords",
                        placeholder="e.g., plastic, solar, water waste, energy saving...",
                        lines=3
                    )
                    generate_tips_btn = gr.Button("Generate Eco Tips")

                with gr.Column():
                    tips_output = gr.Textbox(label="Sustainable Living Tips", lines=15)

            generate_tips_btn.click(eco_tips_generator, inputs=keywords_input, outputs=tips_output)
```

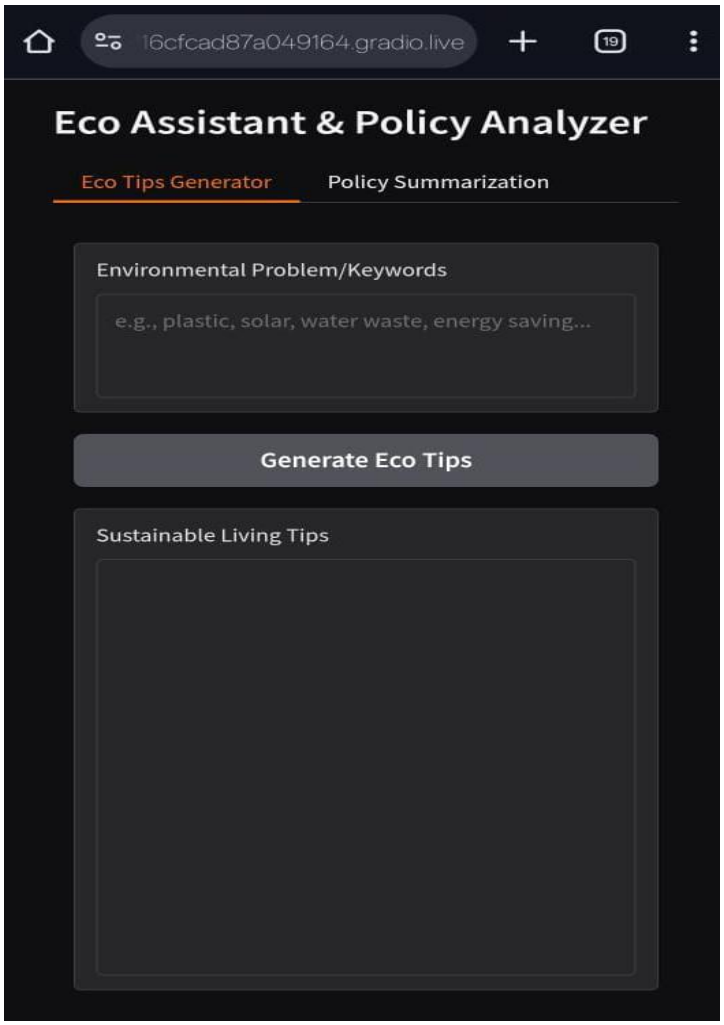
```
    with gr.TabItem("Policy Summarization"):
        with gr.Row():
            with gr.Column():
                pdf_upload = gr.File(label="Upload Policy PDF", file_types=[".pdf"])
                policy_text_input = gr.Textbox(
                    label="Or paste policy text here",
                    placeholder="Paste policy document text...",
                    lines=5
                )
                summarize_btn = gr.Button("Summarize Policy")

            with gr.Column():
                summary_output = gr.Textbox(label="Policy Summary & Key Points", lines=20)

        summarize_btn.click(policy_summarization, inputs=[pdf_upload, policy_text_input],
outputs=summary_output)

app.launch(share=True)
```

OUTPUT:


16cfcad87a049164.gradio.live
19
Eco Assistant & Policy Analyzer
Eco Tips Generator
Policy Summarization
Environmental Problem/Keywords
e.g., plastic, solar, water waste, energy saving...
Generate Eco Tips
Sustainable Living Tips