**Project Title**

**Project Documentation**

**1.Introduction**

Project title: **Smart SDLC - Al-Enhanced Software Development Lifecycle**

Team leader:

**SASI. L**

Team member:

**SURENDHAR. S**

Team member:

**ABDHULLA. S**

Team member:

**MINNAL RAJA. R**

**2.project overview**

**Purpose:**

The purpose of a Smart SDLC (Software Development Life Cycle) project is to revolutionize the way software is planned, developed, tested, and deployed by integrating intelligence, automation, and real-time insights into every stage of the lifecycle. Unlike traditional SDLC models, the Smart SDLC leverages Al-driven tools, cloud platforms, and predictive analytics to streamline processes, reduce risks, and improve collaboration among stakeholders.

For developers, it provides intelligent recommendations, automated code reviews, and continuous integration pipelines. For project managers, it offers smart dashboards, predictive timelines, and resource optimization insights. For stakeholders, it ensures greater transparency, traceability, and faster delivery of high-quality software.

**1.Introduction – Smart SDLC**

The Software Development Life Cycle (SDLC) is a structured process that defines the steps required to design, develop, test, and deploy high-quality software. While traditional SDLC models (such as Waterfall, Spiral, or Agile) provide frameworks for systematic development, they often lack adaptability, automation, and intelligence to address the dynamic needs of today’s fast-paced digital world.

The Smart SDLC Is an evolution of the traditional SDLC. It integrates artificial intelligence, automation, data analytics, and smart collaboration tools into every phase of the lifecycle. This approach not only enhances efficiency but also minimizes risks, optimizes resources, and ensures continuous improvement.

**2. SDLC Model Used**

Model Chosen: Agile SDLC with DevOps integration.

Reason: Agile allows flexibility and continuous iteration while DevOps ensures faster CI/CD pipeline and automated testing/deployment.

Agile Practices Used:

Daily Stand-ups

Sprint Planning

Sprint Review & Retrospective

Continuous Feedback Loop

**3. Phases of SDLC**

**3.1 Requirement Analysis:**

Stakeholder interviews and brainstorming sessions.

Functional Requirements:

User registration/login (future enhancement).

Interactive project showcase. -Contact form with validation.

-3D animations.

Non-Functional Requirements:

Performance: Load under 3 seconds.

-Security: HTTPS, form validation.

-Scalability: Cloud hosting.

**3.2 System Design:**

HLD: Overall architecture with front-end (HTML/CSS/JS) and optional backend (Node.js/Firebase).

* LLD: Detailed component diagrams, navigation flow, state management for interactivity.

**3.3 Implementation:**

* -Tools: VS Code, GitHub, Figma, Postman.
* -Technologies: HTML5, CSS3 (with Tailwind), JavaScript ES6+, optional Node.js, Firebase.
* - Version Control: Git + GitHub.

**3.4 Testing:**

* Levels of Testing:
* Unit Testing: JavaScript functions.
* -Integration Testing: Form submission + database (if enabled).
* System Testing: Portfolio performance across devices.
* Acceptance Testing: Based on user/client feedback.

**3.5 Deployment:**

* -CI/CD pipeline using GitHub Actions.
* -Hosting via Vercel/Netlify.
* -Domain mapped with SSL certificate.
* 3.6 Maintenance:
* -Patch updates (bugs, design fixes).
* Feature upgrades.
* Backup strategy using Git + Cloud storage.

**4.Running the Application**

Once the Smart SDLC (Software Development Life Cycle) application has been developed, tested, and deployed, running the application involves ensuring smooth execution in the target environment. This phase bridges development with real-world usage.

**5.Smart SDLC API Documentatio**

The Smart SDLC API enables developers, city officials, and citizens to interact with the Sustainable Smart City Assistant. It provides secure access to core features such as project management, resource optimization, citizen engagement, and Al-driven insights.

**6. Tools & Technologies**

Frontend: HTML5, CSS3, JavaScript, Tailwind CSS.

Backend (optional): Node.js, Firebase.

Design: Figma, Adobe XD.

Testing: Jest (for JS), BrowserStack (cross-device).

Version Control: Git, GitHub.

Deployment: Netlify, Vercel, GitHub Pages.

Project Management: Jira/Trello for Agile boards.

7.**Smart SDLC (Software Development Life Cycle)**

1. **Dashboard Layout** – left sidebar for stages, main content for details.
2. **SDLC Stages as Interactive Steps** – Requirements → Design → Development → Testing → Deployment → Maintenance.
3. **Progress Visualization** – timeline, progress bar, or radial indicator.
4. **Smart Features**
   * Tooltips with AI-driven suggestions.
   * Collapsible cards for each stage.
   * Dark futuristic theme (glassmorphism / neon glow).

**8.Testing Stage in Smart SDLC**

Goal: Verify the software works correctly, meets requirements, and has no critical bugs.

Key Activities:

Unit testing (checking individual modules).

Integration testing (checking modules work together).

System testing (validating end-to-end).

User Acceptance Testing (UAT).

Automated test scripts for regression.

Tools: Selenium, Junit, Cypress, Jest, Postman, etc.

Outputs: Test reports, bug logs, quality metrics.

**9.program code**

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€}”

Def requirement\_analysis(pdf\_file, prompt\_text):

# Get text from PDF or prompt

If pdf\_file is not None:

Content = extract\_text\_from\_pdf(pdf\_file)

Analysis\_prompt = f”Analyze the following document and extract key software requirements. Organize them into functional requirements, non-functional requirements, and technical specifications:\n\n{content}”

Else:

Analysis\_prompt = f”Analyze the following requirements and organize them into functional requirements, non-functional requirements, and technical specifications:\n\n{prompt\_text}”

Return generate\_response(analysis\_prompt, max\_length=1200)

Def code\_generation(prompt, language):

Code\_prompt = f”Generate {language} code for the following requirement:\n\n{prompt}\n\nCode:”

Return generate\_response(code\_prompt, max\_length=1200)

# Create Gradio interface

With gr.Blocks() as app:

Gr.Markdown(“# AI Code Analysis & Generator”)

With gr.Tabs():

With gr.TabItem(“Code Analysis”):

With gr.Row():

With gr.Column():

Pdf\_upload = gr.File(label=”Upload PDF”, file\_types=[“.pdf”])

Prompt\_input = gr.Textbox(

Label=”Or write requirements here”,

Placeholder=”Describe your software requirements…”,

Lines=5

)

Analyze\_btn = gr.Button(“Analyze”)

With gr.Column():

Analysis\_output = gr.Textbox(label=”Requirements Analysis”, lines=20)

Analyze\_btn.click(requirement\_analysis, inputs=[pdf\_upload, prompt\_input], outputs=analysis\_output)

With gr.TabItem(“Code Generation”):

With gr.Row():

With gr.Column():

Code\_prompt = gr.Textbox(

Label=”Code Requirements”,

Placeholder=”Describe what code you want to generate…”,

Lines=5

)

Language\_dropdown = gr.Dropdown(

Choices=[“Python”, “JavaScript”, “Java”, “C++”, “C#”, “PHP”, “Go”, “Rust”],

Label=”Programming Language”,

Value=”Python”

)

Generate\_btn = gr.Button(“Generate Code”)

With gr.Column():

Code\_output = gr.Textbox(label=”Generated Code”, lines=20)

Generate\_btn.click(code\_generation, inputs=[code\_prompt, language\_dropdown], outputs=code\_output)

App.launch(share=True)

**Output**:



