Software Design Specification

Project Code: Internal Advisor:

**UI/UX Automation Using LLM**

Dr. Fahad Maqbool

External Advisor:

Project Manager:

Dr. Muhammad Ilyas

Project Team:

Muhammad Dawood (BSCS51F21S089) Muhammad Rashid (BSCS51F21S084) Ghulam Rasool (BSCS51F21S080)

Submission Date:

01/02/2025

Project Manager’s Signature

# Document Information

|  |  |
| --- | --- |
| **Category** | **Information** |
| Customer | UI/UX Designers, Design Teams, and Developers. |
| Project | UI/UX Automation Using LLM |
| Document | Software Design Specification |
| Document Version | 1.0 |
| Identifier | PGBH01-2003-DS |
| Status | Draft |
| Author(s) | Muhammad Dawood, Muhammad Rashid, Ghulam Rasool |
| Approver(s) | Dr. Muhammad Ilyas |
| Issue Date | November 26, 2024 |
| Document Location |  |
| Distribution | 1. Dr. Fahad Maqbool 2. Dr. Muhammad Ilyas 3. University of Sargodha |

**Definition of Terms, Acronyms and Abbreviations**

|  |  |
| --- | --- |
| **Term** | **Description** |
| LLM | Large Language Model, used for generating designs and contextual suggestions. |
| GraphRAG | Graph-based Retrieval-Augmented Generation for contextual responses. |
| OWL | Web Ontology Language for semantic consistency. |
| Neo4j | Graph database used for storing and querying UI/UX design knowledge. |
| UI | User Interface. |
| UX | User Experience. |
| Figma | Collaborative design tool for exporting high-fidelity designs. |
|  |  |

**Table of Contents**

1. [Introduction 4](#_bookmark0)
   1. [Purpose of Document4](#_bookmark1)
   2. [Project Overview 4](#_bookmark2)
   3. [Scope 4](#_bookmark3)
2. [Design Considerations 4](#_bookmark4)
   1. [Assumptions and Dependencies 4](#_bookmark5)
   2. [Risks and Volatile Areas 5](#_bookmark6)
3. [System Architecture 5](#_bookmark7)
   1. [System Level Architecture 5](#_bookmark8)
   2. [Sub-System / Component / Module Level Architecture 6](#_bookmark9)
   3. [Sub-Component / Sub-Module Level Architecture (1…n) 6](#_bookmark10)
4. [Design Strategies 6](#_bookmark11)
   1. [Strategy 1…n 6](#_bookmark12)
5. [Detailed System Design 6](#_bookmark13)
6. [References 7](#_bookmark14)
7. [Appendices 8](#_bookmark15)

# Introduction

## Purpose of Document

This document specifies the software design for the **UI/UX Automation Using LLM** project. It establishes a blueprint for the system's development, ensuring adherence to the defined requirements and delivering a scalable and efficient product. The audience includes stakeholders, system architects, developers, and QA teams.

## Project Overview

The project leverages **Large Language Models (LLMs)** to automate UI/UX workflows, enabling seamless wireframe generation, validation, and export. Integration with Neo4j for knowledge graphs and OWL for ontologies ensures adherence to UI/UX standards.

## Key Features:

* + - Automated wireframe generation from natural language descriptions.
    - Context-aware design validation using OWL-defined principles.
    - Export functionality to Figma for high-fidelity design outputs.
    - AI-driven design suggestions based on industry trends.

## Scope

**In-Scope:**

* + - Automating design workflows for UI/UX designers.
    - Providing intuitive feedback for improvement.
    - Knowledge graph management and updates for scalable usage.

## Out-of-Scope:

* Prototypes with interactive elements.
* Domain-specific UI/UX needs outside the ontology's scope.

# Design Considerations

## Assumptions and Dependencies

**Assumptions:**

* + - Users are proficient in basic UI/UX principles and tools like Figma.
    - Internet connectivity will be reliable for LLM and knowledge graph queries.
    - APIs (OpenAI, Figma) remain stable and accessible throughout development.

## Dependencies:

* + - **LLM APIs:** Dependence on services like OpenAI for processing natural language.
    - **Neo4j Database:** Ensures efficient knowledge storage and querying.
    - **Cloud Infrastructure:** Hosting for scalability and consistent performance.
    - **Ontology Updates:** Regular maintenance required for industry-relevant suggestions.

## Risks and Volatile Areas

* + - **Evolving Standards:** Changing UI/UX trends may necessitate updates to ontologies and suggestions.
    - **API Downtime:** Dependence on third-party APIs may lead to disruptions.
    - **Scalability Concerns:** High user demand could affect system performance if not optimized.
    - **Data Security:** Protecting sensitive design data and adhering to regulations like GDPR is critical.

# System Architecture

## System Level Architecture

The system is organized into three primary layers:

## Presentation Layer:

* + - User-facing interface for accessing features.
    - Includes functionalities like design input, wireframe generation, and validation feedback.

**Business Logic Layer:**

* + - Handles core workflows including GraphRAG-based design generation.
    - Interacts with the ontology for design validation.

**Data Layer:**

* + - Manages Neo4j-based knowledge graphs and data persistence.

## Sub-System / Component / Module Level Architecture

**Wireframe Generation Subsystem:**

* + - Converts textual design requirements into structured wireframes using LLM and GraphRAG.

## Design Validation Subsystem:

* + - Validates wireframes against UI/UX principles.
    - Suggests improvements for usability and alignment.

## Feedback Subsystem:

* + - Provides actionable design insights based on contextual queries.

## Export Subsystem:

* + - Enables seamless export of designs to Figma.

## Sub-Component / Sub-Module Level Architecture (1…n)

* + - **Wireframe Generator:** Handles LLM queries and graph-based suggestions.
    - **Validator:** Ensures compliance with UI/UX principles.
    - **Knowledge Graph Manager:** Updates and retrieves data from Neo4j.
    - ***Exporter:*** *Converts wireframes into Figma-compatible formats.*

# Design Strategies

* 1. **Strategy 1…n**
     + **Scalability:** Modular components for future enhancements.
     + **Data Reuse:** Centralized knowledge base with reusable components.
     + **Cross-Platform Accessibility:** Responsive design for web and mobile interfaces.
     + **Performance Optimization:** Asynchronous processing for efficient LLM queries.
     + ***Security Measures:*** *End-to-end encryption and role-based access.*

# Detailed System Design

## Class Diagram:

## 

* + - **UIComponent (abstract base)**
    - **Attributes**

# id: String — unique identifier

# isVisible: Boolean — whether component is drawn

# position: Point — 2D coordinates on screen (e.g. { x: 100, y: 50 })

* + - **Methods**

# render(): actually draws the component if isVisible == true.

# update(): recalculates any internal state (e.g. after property changes).

# handleEvent(evt: Event): default dispatcher that routes events down to specialized handlers (e.g. click, hover).

# InteractiveComponent (inherits UIComponent)

* + - **Attributes**

# isDisabled: Boolean — if true, ignores input events

# tooltip: String — text shown on hover

* + - **Methods**

# onClick(evt: MouseEvent): called by handleEvent when a click occurs within bounds and !isDisabled.

# onHover(evt: MouseEvent): called when the pointer enters/leaves; may show/hide tooltip.

# Interaction: UIComponent.handleEvent checks event type; if evt is click or hover and this is an InteractiveComponent, it down-casts and calls onClick/onHover.

## LayoutComponent (inherits UIComponent)

* + - **Attributes**

# spacing: Integer — pixel gap between children

# alignment: Alignment — e.g. LEFT, CENTER, RIGHT

# children: List<UIComponent> — contained components

* + - **Methods**

# arrange(): computes and sets each child’s position based on spacing & alignment.

# add(child): appends a child and calls arrange().

# remove(child): removes and re-arranges.

# Interaction: When you call layout.arrange(), it iterates over children, invokes each child’s update(), then places them in a grid or stack depending on subclass.

## Button (inherits InteractiveComponent)

* + - **Attributes**

# label: String — text drawn inside the button

* + - **Methods**

# click(): public API to programmatically trigger onClick.

# (Inherited) onClick: typically raises a “clicked” event to application logic.

## TextBox (inherits InteractiveComponent)

* + - **Attributes**

# text: String — current content

* + - **Methods**

# setText(value): updates text and calls update().

# clear(): sets text = "".

## Grid (inherits LayoutComponent)

* + - **Attributes**

# rows: Integer — number of rows

# cols: Integer — number of columns

* + - **Methods**

# getCell(r, c): returns the component at row r, column c (or null).

# Interaction: Grid.arrange() divides its own bounding box into rows × cols cells, places each child in sequence or by explicit row/col metadata.

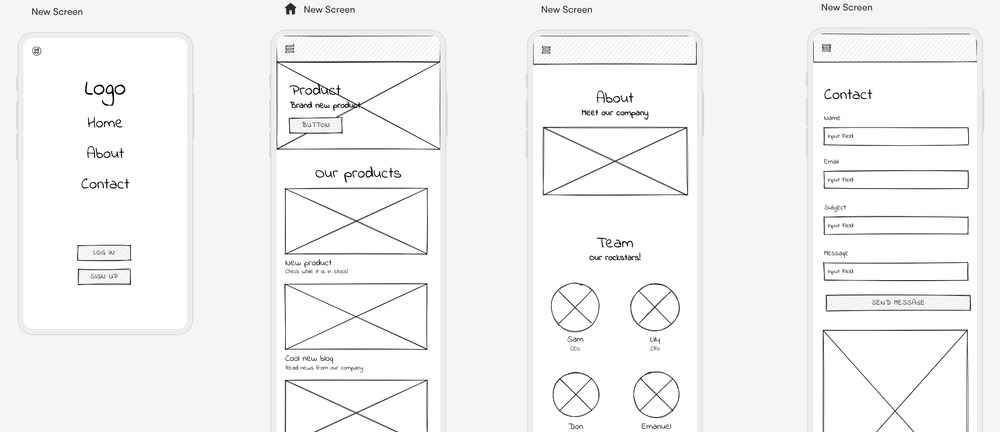
## StackPanel (inherits LayoutComponent)

* + - **Attributes**

# orientation: Orientation — either HORIZONTAL or VERTICAL

# Interaction: StackPanel.arrange() lines children up in a row or column, spacing them by spacing, and aligning them according to alignment.

## Sequence Diagram:



This sequence diagram lays out the step-by-step flow for automatically generating a styled wireframe and pushing it into Figma:

## Prompt & Context

* + - The Designer starts by sending a natural-language prompt (e.g. “Recipe page for food app”) to the Wireframe Generation Service.

## Raw Wireframe Generation

* + - The Wireframe Service invokes its fine-tuned LLM (trained on UI patterns) to produce a raw wireframe model—a structured JSON or UI tree defining containers, text blocks, image slots, and interactive elements.

## Beautification

* + - The Designer then calls the Beautify Engine, supplying the raw model plus any chosen design tokens or platform rules.
    - The engine consults its styling rules (spacing scales, typography, color palettes) and returns a polished mid-fidelity mockup in vector or HTML/CSS form.

## Export to Figma

* + - With the styled screens approved, the Designer triggers the Figma Exporter, sending over the vector assets along with layer and component metadata.
    - The exporter uses the Figma API (or a plugin) to recreate each screen as native Figma frames, groups, and symbols.

## Completion

* + - Figma responds with confirmation and provides a shareable link (or embeds the new pages in the existing project), making the wireframes instantly available for collaborative editing, annotation, and hand-off.

## State Transition Diagram:

## 

**Input stage:**

* + - The "owlThing" node on the far left represents the entry point where design requirements would be processed into the system.

**Generated stage**:

* + - The "DesignPrinciple" central node represents the core principles that the LLM would apply when generating wireframes from user input.

**Validated stage**:

* + - This is where the diagram is most relevant, showing the extensive validation framework:
    - Major principles like Visibility, Consistency, Responsiveness, Accessibility, and Usability branch out
    - Each major principle further connects to specific sub-principles (is-a relationships)
    - For example, Visibility connects to Emphasis, Unity, Contrast, Rhythm, and Balance
    - Accessibility connects to Understandability, Robustness, Perceivable, and Operable

**Exported stage**:

* + - While not explicitly shown, the wireframe would only proceed to export after validation against this comprehensive set of principles.

## Logical Data Model (E/R Diagram):

## 

**Entities:**

* + - **User**
    - Represents individuals who interact with the system.
    - Likely attributes: UserID, Name, Email, etc.

**Wireframe**

* + - Represents design artifacts or layout structures created within the system.
    - Likely attributes: WireframeID, Title, CreationDate, etc.

**Ontology**

* + - Represents a structured set of rules or semantic constraints used to validate wireframes.
    - Likely attributes: OntologyID, RuleName, Description, etc.

**Relationships:**

**User "generates" Wireframes**

* + - A one-to-many relationship: each user can generate multiple wireframes.
    - Typically implemented by associating UserID as a foreign key in the Wireframe entity.

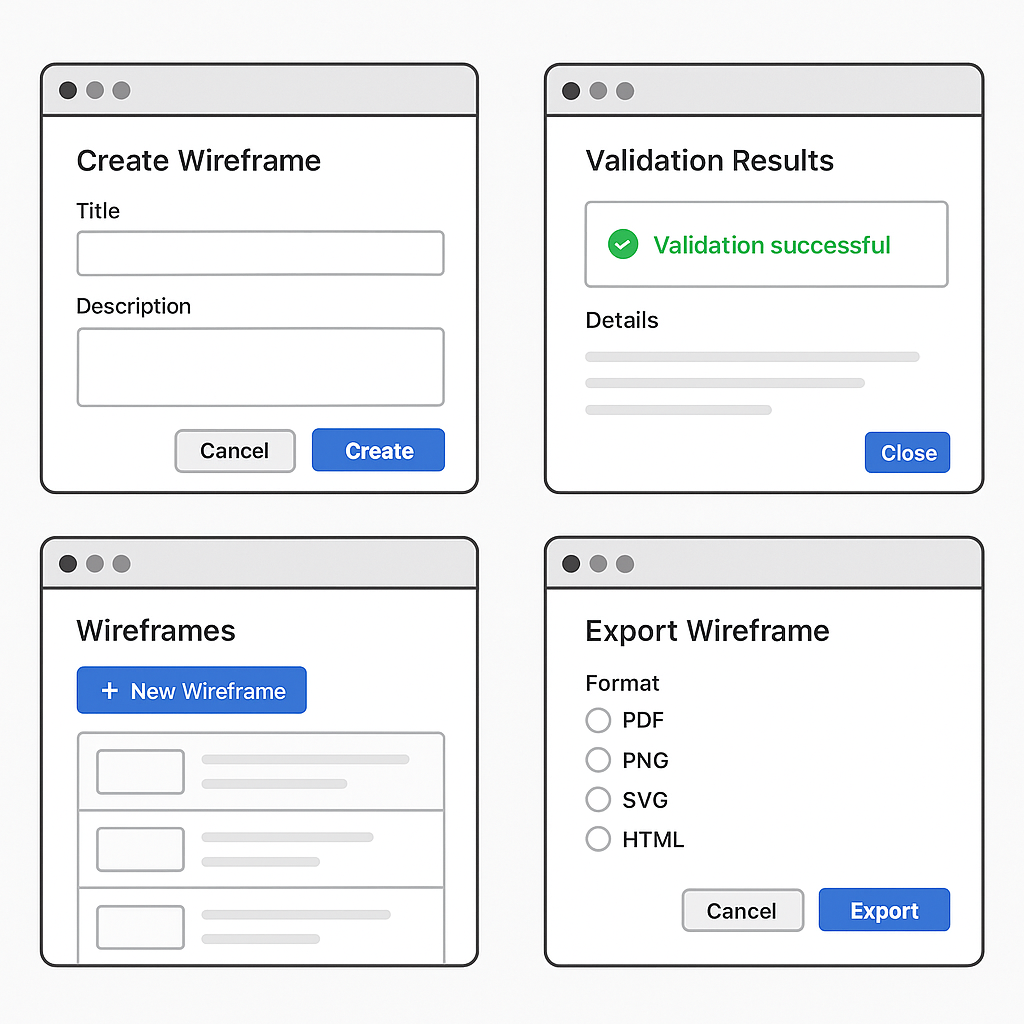
**Wireframes "validated by" Ontology**

* + - A many-to-many relationship: each wireframe can be validated by multiple ontology rules, and each rule can validate multiple wireframes.
    - This could be represented logically by a linking entity or join table in physical design.

## Physical Data Model:

## 

## GUI Designs:



# References

|  |  |  |  |
| --- | --- | --- | --- |
| **Ref. No.** | **Document Title** | **Date of Release/ Publication** | **Document Source** |
| RAG202 3 | GraphRAG Explained: Enhancing RAG with Knowledge Graphs | 2023 | Medium (<https://medium.com/@zilliz_learn/graphrag-explained-enhancing-rag-with-knowledge-graphs-3312065f99e1> |
| OWL200 4 | OWL Reference | Feb 10, 2004 | W3C  ([https://www.w3.org/](https://www.w3.org/TR/owl-ref/) [TR/](https://www.w3.org/TR/owl-ref/) [owl-ref/](https://www.w3.org/TR/owl-ref/)) |
| AIUX202 3 | Artificial Intelligence (AI) for User Experience (UX)  Design | Aug 2023 | Information Technology and People Journal |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Appendices

Appendix

A:Glossary

Appendix B: Workflow Diagrams