****

**COMSATS University Islamabad (CUI) Attock Campus**

Software Requirement Specification  
(SRS DOCUMENT)

for

**Wheat Shield**

Version 1.0

***By***

**Kashif Hussain CIIT/FA20-BCS-019/ATK**

**Muhammad Zubair CIIT/FA20-BCS-041/ATK**

***Supervisor*Mr. Muhammad Wasim Khan**

*Bachelor of Science in Computer Science (2020-2024)*

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason for Changes** | **Version** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

Application Evaluation History

|  |  |
| --- | --- |
| **Comments (by committee)**  **\*include the ones given at scope time both in doc and presentation** | **Action Taken** |
|  |  |
|  |  |
|  |  |
|  |  |
|  |  |

Supervised by

Mr. Muhammad Wasim Khan

Signature\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Table of Contents**

[1. Introduction 1](#_Toc82692022)

[1.1 Purpose 1](#_Toc82692023)

[1.2 Scope 1](#_Toc82692024)

[1.3 Modules 1](#_Toc82692025)

[1.4 Overview 1](#_Toc82692026)

[2. Overall Description 1](#_Toc82692027)

[2.1 Product Perspective 1](#_Toc82692028)

[2.2 User classes and characteristics 1](#_Toc82692029)

[2.3 Operating Environment 1](#_Toc82692030)

[2.4 Design and Implementation Constraints 2](#_Toc82692031)

[3. Requirement Identifying Technique 2](#_Toc82692032)

[4. Functional Requirements 2](#_Toc82692033)

[4.1 Functional Requirement X 3](#_Toc82692034)

[5. Non-Functional Requirements 3](#_Toc82692035)

[5.1 Reliability 3](#_Toc82692036)

[5.2 Usability 4](#_Toc82692037)

[5.3 Performance 4](#_Toc82692038)

[5.4 Security 4](#_Toc82692039)

[6. External Interface Requirements 4](#_Toc82692040)

[6.1 User Interfaces Requirements 4](#_Toc82692041)

[6.2 Software interfaces 5](#_Toc82692042)

[6.3 Hardware interfaces 5](#_Toc82692043)

[6.4 Communications interfaces 5](#_Toc82692044)

[7. References 5](#_Toc82692045)

# 

# Introduction

The introduction of the **Software Requirements Specification** (SRS) for **WheatShield** offers a comprehensive overview of this groundbreaking agricultural solution. This document serves as a detailed guide, encapsulating the purpose, scope, modules, and essence of the entire project. **WheatShield** represents a significant leap in agricultural technology, utilizing advanced machine learning and real-time data analysis to address the critical issue of disease detection in wheat crops. As we delve deeper into this SRS, we will explore the intricacies of the project's backend model training, API implementation, frontend design, and various additional features. This document aims to provide a clear understanding of the innovative approach and meticulous design behind **WheatShield**, empowering readers with a profound insight into the project's functionality and significance.

## Purpose

The purpose of WheatShield is twofold: firstly, to transform agricultural practices by providing farmers and agricultural professionals with an advanced disease detection system. Utilizing cutting-edge machine learning algorithms, WheatShield enables early and precise identification of wheat leaf diseases, allowing for timely interventions and minimizing crop losses. Secondly, WheatShield offers comprehensive weather monitoring capabilities, delivering real-time data on temperature, humidity, wind speeds, and visibility. By merging disease detection and weather monitoring functionalities, the system equips users with a holistic understanding of their agricultural environment. This holistic approach empowers users to make informed decisions, optimize crop management strategies, and embrace sustainable farming practices, leading to enhanced crop yields and economic prosperity within the agricultural sector.

## Scope

The scope of WheatShield encompasses a multifaceted approach to modern agriculture, integrating advanced disease detection and real-time weather monitoring capabilities. Beyond its primary focus on detecting wheat leaf diseases, the system provides users with essential weather-related insights, including temperature, humidity, wind speed, and visibility. This comprehensive approach enables agricultural professionals to make data-driven decisions, aligning their strategies with both crop health and environmental conditions. By combining cutting-edge disease detection algorithms with in-depth weather analysis, WheatShield offers a holistic solution for farmers, fostering sustainable practices and higher yields.

## Modules

WheatShield integrates essential modules to revolutionize wheat farming and disease management:

* **Disease Detection**: Utilizing cutting-edge image processing and machine learning algorithms, this module swiftly identifies stripe rust and septoria diseases in wheat crops. Real-time image analysis enables prompt disease detection, empowering farmers to take immediate corrective actions.
* **Cause Identification**: In-depth analysis of environmental factors, such as weather conditions and soil quality, enables the identification of disease triggers. By pinpointing root causes, agricultural professionals gain valuable insights, enhancing their ability to implement effective preventive measures.
* **Solution Recommendation**: Building on cause analysis, this module generates personalized recommendations and guidelines. Tailored solutions empower users to proactively combat diseases and prevent their recurrence. Precise recommendations optimize disease management efforts, promoting healthier crop growth.
* **Medicine Suggestion**: Accessing a comprehensive database of approved agricultural medicines, this module provides precise recommendations for treating stripe rust and septoria diseases. Appropriate medicine selection ensures targeted disease management, enabling farmers to optimize their resources.
* **Weather Monitoring**: This vital module tracks weather patterns in real-time, providing crucial data for disease prediction and prevention. By analyzing weather conditions, farmers can anticipate disease outbreaks and implement proactive measures, safeguarding their crops effectively.

## Overview

The WheatShield Software Requirements Specification (SRS) meticulously outlines the project's components, functionality, and constraints. This comprehensive document is structured to facilitate a deep understanding of the WheatShield project. The SRS is divided into several sections, including an introduction, system overview, detailed functional and non-functional requirements, system constraints, and user interfaces. By encapsulating critical information such as scope, objectives, and module intricacies, the document guarantees a shared comprehension among all stakeholders. With a focus on clarity and precision, the SRS ensures that the project's requirements and deliverables are clearly defined, serving as a guiding beacon for all project participants.

# Overall Description

In this section, we provide a comprehensive portrayal of the WheatShield system, delving into its high-level overview, the surrounding environment, its user base, and the inherent constraints, assumptions, and dependencies. By examining these key aspects, we gain a holistic understanding of the software's context and the ecosystem in which it operates. This detailed overview sets the stage for a nuanced exploration of the system's intricacies, ensuring a profound comprehension of its functionalities and user interactions.

## Product Perspective

WheatShield stands as an innovative solution in the agricultural technology domain, representing an entirely new product designed to transform wheat farming practices. Unlike its predecessors, WheatShield integrates advanced technologies, including image processing, machine learning, and real-time weather monitoring, to empower farmers and agriculture professionals effectively. It not only detects and manages wheat diseases but also includes a unique feature: a nearest shop recommendation module for medicine procurement

## User classes and characteristics

WheatShield caters to a diverse range of users, each with specific roles and characteristics tailored to their agricultural needs. The primary user classes include:

**1.Farmers**:

* **Characteristics**: Farmers are the end-users of WheatShield, ranging from small-scale farmers to large agricultural enterprises. They possess varying levels of technological expertise.
* **Interactions**: Farmers utilize the mobile or web application to capture images of wheat crops, receive disease diagnosis, access tailored solutions, and receive weather forecasts.

**2.Agriculture Professionals**:

* **Characteristics**: Experts in the agriculture field, such as agronomists and researchers, possessing in-depth knowledge of crop diseases and agricultural practices.
* **Interactions**: Agriculture professionals engage with WheatShield to analyze disease patterns, contribute insights, and collaborate with farmers for disease management strategies.

**3.Medicine Suppliers**:

* **Characteristics**: Local pharmacies, clinics, or authorized suppliers of agricultural medicines.
* **Interactions**: Medicine suppliers are integrated into WheatShield to receive requests for recommended medicines and facilitate seamless procurement for farmers.

**4.System Administrators**:

* **Characteristics**: IT professionals responsible for system maintenance, security, and overall management of WheatShield.
* **Interactions**: System administrators manage user accounts, ensure data security, and address technical issues, ensuring the platform's smooth operation.

## Operating Environment

The software will operate in an environment that includes both hardware and software components. The hardware platform may consist of computers or mobile devices used by farmers, agronomists, and researchers. The software will be compatible with various operating systems such as Windows and this software would be web based. The web application will be hosted on a web server and can be accessed through a web browser. The application will be compatible with modern web browsers such as Google Chrome, Firefox, and Safari.

## Design and Implementation Constraints

*CON-1: The system shall use the OpenCV (Open Source Computer Vision Library) for image processing.*

*CON-2: The application must utilize the Google Maps Geocoding API for geolocation services*

*CON-3: All textual data used by the application shall be stored in JSON format.*

*CON-4: The development shall be carried out using the React Native framework.*

*CON-5: The application shall be developed for both Android and iOS mobile platforms*

# Requirement Identifying Technique

This section describes the requirements identifying technique(s) which further help to derive functional requirements specification. The selection of the technique(s) will depend on the type of project. For instance,

* **Use case (use case diagram):**
* **Use case (detailed use case):**

**Use Case Name: Disease Detection**

**Actor: User**

Description: The "Disease Detection" use case allows users to capture images of wheat crops and uses image processing techniques and machine learning algorithms to detect the presence of stripe rust and septoria diseases in the crops.

**Flow of Events:**

1. The user initiates the "Disease Detection" use case from the application's main menu.
2. The system opens the smartphone's camera viewfinder to allow the user to capture an image or to upload th image.
3. The user captures an image of a wheat crop by tapping the "Capture" button.
4. The system processes the captured image using image processing and machine learning algorithms.
5. The system analyzes the image and identifies the presence of stripe rust and septoria diseases.
6. If diseases are detected, the system displays the results, including the type of disease and its severity, on the user's screen.
7. The user can review the results, assess the severity, and decide on the next steps, such as accessing recommendations or medicine suggestions.

**Alternative Flow:**

If no diseases are detected, the system displays a message indicating that the crop appears healthy, and the use case ends.

**Postconditions:**

The system saves the detection results for future reference.

The user may choose to access further information or recommendations based on the detected diseases.

* **Event- response :**

|  |  |  |  |
| --- | --- | --- | --- |
| **Response** | **Event** | **System State** | **Response** |
| Detect and identify stripe rust and septoria diseases. | User requests detection | Idle | 1. Receive and process the uploaded crop image using image processing and machine learning algorithms. |
| Display detection results, including disease type and severity, to the user. | Cause Identification , User requests cause identification | Cross-traffic signal is green. | 1. Analyze weather conditions, soil quality, and crop management practices based on available data. |
| Identify potential causes of stripe rust and septoria diseases. |  |  |  |
| Display causal factors and recommendations to the user, including suggested actions and preventive measures. | Solution Recommendation | User requests solution recommendations | Idle | 1. Based on identified causes and disease type, provide tailored recommendations. |
| Display a list of recommended medicines to the user | Nearest Shop Recommendation | User requests shop recommendations | Idle | 1. Utilize geolocation data to identify the user's location. |
| Display recommended actions to the user. | Medicine Suggestion | User requests medicine suggestions | Idle |  |
| Find the nearest shops or pharmacies where the suggested medicines are available. | User requests shop recommendations | Idle | 1. Utilize geolocation data to identify the user's location. |
| Display shop details, including addresses and contact information, to the user. | User selects a specific | Shop Selected | 1. Display detailed shop information, including the shop's name, exact address, phone number, hours of operation, and any other relevant contact details. |

* **Storyboarding**:

The storyboard illustration for seamless user experience within the "Disease Detection" module of the WheatShield application. It comprises several key scenes:

**Scene 1: User Interaction**

Image: A user holding a smartphone with the "Wheat Shield" Webapp open.

Action: The user selects the "Services" feature from the web app's nav bar.

**Scene 2: Image Selection**

Image: Smart screen displaying the option to drag and drop an image from the gallery.

Action: User selects an image from the gallery and drops it into the designated area, initiating the disease detection process.

**Scene 3: Image Processing**

Image: The selected image of the wheat crop.

Action: The app processes the image in real-time, and the screen shows the analysis.

**Scene 4: Detection Results**

Image: The smart screen displays the results.

Action: The "Wheat Shield" app successfully detects stripe rust and septoria diseases in the crop. The screen shows a visual overlay on the image indicating the affected areas and provides a disease severity score.

**Scene 5: Recommendations**

Image: The smartphone screen displays disease-specific recommendations.

Action: The app offers recommendations such as treatment options, suggested actions, preventive measures based on the disease type .

**Scene 6: Next Steps**

Image: The user interacting with the recommendations.

Action: The user can tap on the recommendations to view more details, access information about medicines, or find the nearest shops to purchase the suggested remedies.

# Functional Requirements

This section describes the functional requirements of the system expressed in the natural language style. This section is typically organized by feature as a system feature name and specific functional requirements associated with this feature. It is just one possible way to arrange them. Other organizational options include arranging functional requirements by use case, process flow, mode of operation, user class, stimulus, and response depend on what kind of technique has been used to understand functional requirements. Hierarchical combinations of these elements are also possible, such as use cases within user classes. For further detail see Chapter 10 “Documenting the requirements”. Let consider the feature scheme as an example.

## Functional Requirement X

FR-1: The system detect stripe rust and septoria diseases in wheat crops.

FR-2: The system identifies reasons/causes of the detected diseases.

FR-3: The system provides possible solutions or recommendations to mitigate the detected diseases.

FR-4: The system suggests appropriate medicines or treatments for the detected diseases.

FR-5: The system recommends the nearest shop or supplier where the suggested medicines can be purchased.

FR-6: The system is web-based, accessible through a web browser.

Table 1: Description of FR

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Identifier** | **FR-1** | **FR-2** | **FR-3** | **FR-4** | **FR-5** | **FR-6** |
| Title | Disease Detection | Cause Identification | Solution Recommendation | Medicine Suggestion | Nearest Shop Recommendation | System Accessibility |
| Requirement | The system shall detect stripe rust and septoria diseases in wheat crops. | The system shall identify reasons/causes of the detected diseases. | The system shall provide possible solutions or recommendations to mitigate the detected diseases. | The system shall suggest appropriate medicines or treatments for the detected diseases. | The system shall recommend the nearest shop or supplier where the suggested medicines can be purchased. | The system is web-based, accessible through a web browser. |
| Source | Agricultural Experts, Project Stakeholders | Agricultural Experts, Project Stakeholders | Agricultural Experts, Project Stakeholders | Medical Professionals, Project Stakeholders | Geolocation Services, Project Stakeholders | Project Stakeholders, Development Team |
| Rationale | To help farmers identify diseases early for crop protection. | To assist users in understanding the causes behind disease occurrences and taking preventive measures. | To offer guidance and recommendations for effective disease mitigation. | To facilitate quick access to appropriate medicines for disease treatment. | To provide convenient access to the required medicines or treatments. | To make the system accessible to a broader audience through the web. |
| Business Rule (if required) | N/A | N/A | N/A | N/A | N/A | N/A |
| Dependencies | None | FR-1, FR-2 | FR-2 | FR-1 | FR-4 | None |
| Priority | High | High | High | High | High | High |

# Non-Functional Requirements

## Reliability

**Requirement:**

The system shall have a mean time between failures (MTBF) of at least 1000 hours.

**Definition of Failure:**

A failure is defined as any unexpected system crash or unresponsive behavior.

**Consequences of Failure:**

In the event of system failure, users may not be able to perform disease detection, access recommendations, or medicine suggestions.

**Error Detection Strategy:**

The system will implement automated error detection mechanisms to identify and log failures.

**Error Correction Strategy:**

Upon detection of a failure, the system will initiate an automatic recovery process. If automatic recovery fails, an error notification will be sent to the support team..

## Usability

* The system is easy to learn and use, with intuitive user interfaces.
* The system provide error avoidance and recovery mechanisms.
* It ensures efficient interactions and accessibility for users.

## 

## Performance

Performance Requirement (PER-1): The system will complete 95% of disease detection processes within 10 seconds from the time the user submits an image.

Performance Requirement (PER-2): The system will support concurrent access by up to 100 users without a noticeable decrease in response time.

Performance Requirement (PER-3): The system's web-based interface shall load within 5 seconds on a 3G mobile network.

Performance Requirement (PER-4): Recommendations and medicine suggestions shall be generated and displayed to the user within 5-10 seconds of disease detection.

Performance Requirement (PER-5): The system shall maintain a 99.9% uptime, allowing scheduled maintenance for a maximum of 1 hour per week.

## Security

* The system protects sensitive data and ensure the security of the software.
* It focuses on measures to prevent unauthorized access and protect against potential vulnerabilities.

# External Interface Requirements

## User Interfaces Requirements

**GUI Standards and Style Guides:**

The user interfaces of the Wheat Shield system should adhere to established GUI (Graphical User Interface) standards or internal product family style guides to maintain a consistent and professional appearance.

## Software interfaces

**SI-1: Disease Detection Module**

SI-1.1: The Disease Detection module shall interface with image processing libraries (e.g., OpenCV) for image analysis.

SI-1.2: The Disease Detection module shall integrate with machine learning libraries (e.g., TensorFlow) for disease detection using trained models.

**SI-2: Cause Identification Module**

SI-2.1: The Cause Identification module shall interface with weather data APIs to retrieve real-time weather conditions.

SI-2.2: The Cause Identification module shall connect to soil quality databases to access soil data.

SI-2.3: The Cause Identification module shall gather crop management data through manual input or integration with farming management systems.

**SI-3: Solution Recommendation Module**

SI-3.1: The Solution Recommendation module shall connect to the Cause Identification module to access identified causes.

SI-3.2: It may provide recommendations through a user interface and may offer links to relevant resources or guidelines.

**SI-4: Medicine Suggestion Module**

SI-4.1: The Medicine Suggestion module will access a database of approved medicines for stripe rust and septoria diseases.

SI-4.2: It may integrate with external sources to stay updated on the availability of new medicines.

**SI-5: Nearest Shop Recommendation Module**

SI-5.1: The Nearest Shop Recommendation module shall utilize geolocation data and may interact with mapping services (e.g., Google Maps).

SI-5.2: It shall provide users with recommendations for nearby shops where suggested medicines can be purchased.

## Hardware interfaces

## Hardware Device: Web Browsers

## Supported Device Types:

Desktop web browsers (e.g., Chrome, Firefox, Edge)

Mobile web browsers (e.g., Chrome for Android)

**Purpose:**

Web browsers are the client-side software components used by end-users to access and interact with the Wheat Shield website. The purpose of this hardware component is to render and display the website's user interface, process user input, and interact with the server-side components of the website. Users access the Wheat Shield website through various web browsers running on different devices, such as desktop computers, laptops, mobile phones, and tablets. The website's design and functionality must be compatible with and responsive to the diverse range of web browsers to ensure a consistent and user-friendly experience for all users.

**Data Interactions:**

**Input:** Web browsers receive user input, such as mouse clicks, keyboard input, and touchscreen gestures, and transmit this input to the website for processing.

**Output:** Browsers render and display the website's content, including text, images, videos, and interactive elements.

**Control Interactions:**

**Input:** Users interact with the website's controls (e.g., buttons, forms) using their web browsers to initiate actions or submit data.

**Output:** The website's code may execute scripts within the browser to control aspects of the user interface and website behavior.

**Communication Protocols:**

Websites typically communicate with web browsers over HTTP and HTTPS protocols. This includes data retrieval, form submissions, and client-server interactions.

**Data Format:**

Data exchanged between the website and the web browser is typically in HTML, CSS, JavaScript, and other web-compatible formats.

**Error Handling:**

Web browsers may display error messages or warnings to users in case of issues, such as broken links, unresponsive scripts, or security-related concerns.

**Dependencies:**

The website may rely on standard web technologies and libraries (e.g., jQuery, Bootstrap) for optimal compatibility with web browsers.

## Communications interfaces

CI-1: The Disease Detection Module may communicate with external image capture devices, such as cameras, through compatible APIs or drivers.

CI-2: The Nearest Shop Recommendation Module shall communicate with geolocation services and mapping APIs to provide location-based recommendations.

CI-3: Users may receive notifications or alerts via email or SMS based on recommendations and updates from the software modules.

CI-4: The system may communicate with external databases or online resources to update its knowledge base, such as the approved medicines database.

# References

* 1. Nielsen, J. (2000). Designing Web Usability: The Practice of Simplicity. New Riders.
  2. W3C. (n.d.). Cascading Style Sheets (CSS). World Wide Web Consortium. [https://www.w3.org/Style/CSS/Overview.en.html
  3. Wheat Shield Project Style Guide. (2023). Internal document outlining the style and design guidelines for the Wheat Shield website.
  4. Wheat Shield Database Schema. (2023). Internal document specifying the structure and relationships within the Wheat Shield database.

**Appendix A**

**Example: Context Diagram**

The Cafeteria Ordering System is a new software system that replaces the current manual and telephone processes for ordering and picking up meals in the Process Impact cafeteria. The context diagram in Figure 1 illustrates the external entities and system interfaces.

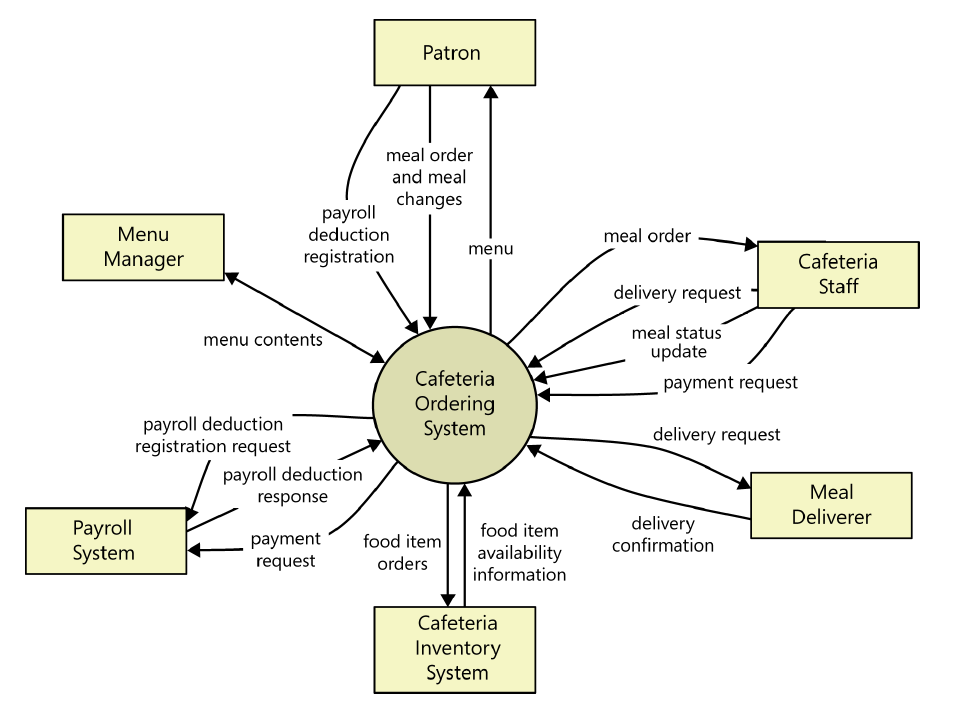


Figure 1: Context diagram of the Cafeteria Ordering System.

**Example: User Classes and Characteristics**

Table 1 Shows user classes and characteristic for Cafetaria ordering system

|  |  |
| --- | --- |
| **User class** | **Description** |
| **Patron** | A Patron is a Process Impact employee who wants to order meals to be delivered from the company cafeteria. There are about 600 potential Patrons, of which 300 are expected to use the COS an average of 5 times per week each. Patrons will sometimes order multiple meals for group events or guests. An estimated 60 percent of orders will be placed using the corporate intranet, with 40 percent of orders being placed from home or by smartphone or tablet apps. |
| **Cafeteria Staff** | The Process Impact cafeteria employs about 20 Cafeteria Staff who will receive orders from the COS, prepare meals, package them for delivery, and request delivery. Most of the Cafeteria Staff will need training in the use of the hardware and software for the COS. |
| **Menu Manager** | The Menu Manager is a cafeteria employee who establishes and maintains daily menus of the food items available from the cafeteria. Some menu items may not be available for delivery. The Menu Manager will also define the cafeteria’s daily specials. The Menu Manager will need to edit existing menus periodically. |
| **Meal Deliverer** | As the Cafeteria Staff prepare orders for delivery, they will issue delivery requests to a Meal Deliverer’s smartphone. The Meal Deliverer will pick up the food and deliver it to the Patron. A Meal Deliverer's other interactions with the COS will be to confirm that a meal was (or was not) delivered. |

**Example: Use case Diagram**

Following use case diagram is of an appointment system in which all the use case diagram relationships are presented. In further in **Table 2** to the detail of use case diagram sytax is provided.

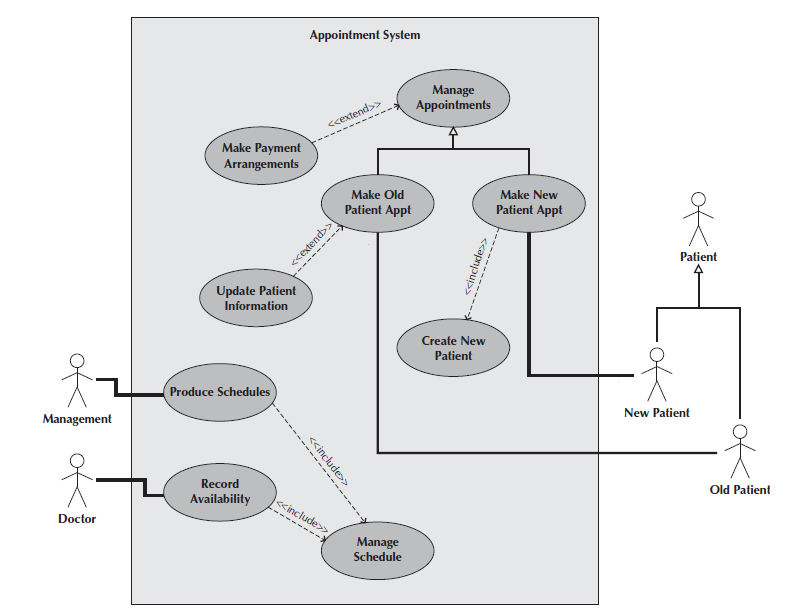


Figure 2: Use Case Diagram of an Appointment System

**Syntax for Use case Diagram**

Table 2 Syntax for Use case Diagram

|  |  |
| --- | --- |
| **An actor:**  ■ Is a person or system that derives benefit from and is external to the subject.  ■ Is depicted as either a stick figure (default) or, if a nonhuman actor is involved, a  rectangle with <<actor>> in it (alternative).  ■ Is labeled with its role.  ■ Can be associated with other actors using a specialization/superclass association,  denoted by an arrow with a hollow arrowhead.  ■ Is placed outside the subject boundary. |  |
| **A use case:**  ■ Represents a major piece of system functionality.  ■ Can extend another use case.  ■ Can include another use case.  ■ Is placed inside the system boundary.  ■ Is labeled with a descriptive verb–noun phrase. |  |
| **subject boundary:**  ■ Includes the name of the subject inside or on top.  ■ Represents the scope of the subject, e.g., a system or an individual  business process. |  |
| **An association relationship:**  ■ Links an actor with the use case(s) with which it interacts. |  |
| **An include relationship:**  ■ Represents the inclusion of the functionality of one use case within another.  ■ Has an arrow drawn from the base use case to the used use case. |  |
| **An extend relationship:**  ■ Represents the extension of the use case to include optional behavior.  ■ Has an arrow drawn from the extension use case to the base use case. |  |
| **A generalization relationship:**  ■ Represents a specialized use case to a more generalized one.  ■ Has an arrow drawn from the specialized use case to the base use case. |  |

**Detail Use Case Example**

The **Table 3** below indicate a comprehensive use case template filled in with an example drawn from the Cafeteria ordering system (COS).

Table 3 Show the detail use case template and example

|  |  |
| --- | --- |
| **Use Case ID:** | Enter a unique numeric identifier for the Use Case. e.g. UC-1 |
| **Use Case Name:** | Enter a short name for the Use Case using an active verb phrase. e.g.  Order a Meal |
| **Actors:** | [An actor is a person or other entity external to the software system being specified who interacts with the system and performs use cases to accomplish tasks.] e.g.   |  |  |  |  | | --- | --- | --- | --- | | Primary Actor: | Patron | Secondary Actors: | Cafeteria Inventory System | |
| **Description:** | [Provide a brief description of the reason for and outcome of this use case.] e.g.  A Patron accesses the Cafeteria Ordering System from either the corporate intranet or external Internet, views the menu for a specific date, selects food items, and places an order for a meal to be picked up in the cafeteria or delivered to a specified location within a specified 15-minute time window. |
| **Trigger:** | [Identify the event that initiates the use case.]e.g.  A Patron indicates that he wants to order a meal. |
| **Preconditions:** | [List any activities that must take place, or any conditions that must be true, before the use case can be started.  PRE-1. Patron is logged into COS.  PRE-2. Patron is registered for meal payments by payroll deduction. |
| **Postconditions:** | [Describe the state of the system at the conclusion of the use case execution.  POST-1. Meal order is stored in COS with a status of “Accepted.”  POST-2. Inventory of available food items is updated to reflect items in this order.  POST-3. Remaining delivery capacity for the requested time window is updated. |
| **Normal Flow:** | [Provide a detailed description of the user actions and system responses that will take place during execution of the use case under normal, expected conditions.  1.0 Order a Single Meal  1. Patron asks to view menu for a specific date. (see 1.0. E1, 1.0.E2)  2. COS displays menu of available food items and the daily special.  3. Patron selects one or more food items from menu. (see 1.1)  4. Patron indicates that meal order is complete. (see 1.2)  5. COS displays ordered menu items, individual prices, and total price, including taxes and delivery charge.  6. Patron either confirms meal order (continue normal flow) or requests to modify meal order (return to step 2).  7. COS displays available delivery times for the delivery date.  8. Patron selects a delivery time and specifies the delivery location.  9. Patron specifies payment method.  10. COS confirms acceptance of the order.  11. COS sends Patron an email message confirming order details, price, and delivery instructions.  12. COS stores order, sends food item information to Cafeteria Inventory System, and updates available delivery times. |
| **Alternative Flows:** | [Document legitimate branches from the main flow to handle special conditions (also known as extensions). For each alternative flow reference the branching step number of the normal flow and the condition which must be true for this extension to be executed. e.g.  1.1 Order multiple identical meals  1. Patron requests a specified number of identical meals. (see 1.1. E1)  2. Return to step 4 of normal flow.  1.2 Order multiple meals  1. Patron asks to order another meal.  2. Return to step 1 of normal flow.  Note: Insert a new row for each distinctive alternative flow. ] |
| **Exceptions:** | 1.0. E1 Requested date is today and current time is after today’s order cutoff time  1. COS informs Patron that it’s too late to place an order for today.  2a. If Patron cancels the meal ordering process, then COS terminates use case.  2b. Else if Patron requests another date, then COS restarts use case.  1.0. E2 No delivery times left  1. COS informs Patron that no delivery times are available for the meal date.  2a. If Patron cancels the meal ordering process, then COS terminates use case.  2b. Else if Patron requests to pick the order up at the cafeteria, then continue with normal flow, but skip steps 7 and 8.  1.1. E1 Insufficient inventory to fulfill multiple meal order  1. COS informs Patron of the maximum number of identical meals he can order, based on current available inventory.  2a. If Patron modifies number of meals ordered, then return to step 4 of normal flow.  2b. Else if Patron cancels the meal ordering process, then COS terminates use case. |
| **Business Rules** | Use cases and business rules are intertwined. Some business rules constrain which roles can perform all or parts of a use case. Perhaps only users who have certain privilege levels can perform specific alternative flows. That is, the rule might impose preconditions that the system must test before letting the user proceed. Business rules can influence specific steps in the normal flow by defining valid input values or dictating how computations are to be performed e.g.  BR-1 Delivery time windows are 15 minutes, beginning on each quarter hour.  BR-2 Deliveries must be completed between 11:00 A.M. and 2:00 P.M. local time, inclusive.  Note: If you are maintaining the business rule in a separate table in SRS then only mention here their IDs. |
| **Assumptions:** | [List any assumptions.   1. e.g. Assume that 15 percent of Patrons will order the daily special (Source: previous 6 months of cafeteria data). |

Event-Response Tables

In order to develop Event Response table fisrt it is required to identify all possible events. Following is an example of events list and event-response table of higway intersection system.

The highway intersection system described earlier has to deal with various events, including these:

* A sensor detects a car approaching in one of the through lanes.
* A sensor detects a car approaching in a left-turn lane.
* A pedestrian presses a button to request to cross a street.
* One of many timers counts down to zero.

**Table 4** presents a fragment of what an event-response table might look like for such a system. Each expected system behavior consists of a combination of event, system state, and response.

**Note:** You may add more information to event-response table in order to perform detail requirement analysis which includes:

* The event frequency (how many times the event takes place in a given time period, or a limit to how many times it can occur).
* Data elements that are needed to process the event.
* The state of the system after the event responses are executed.

Table 4 Partial Event-Response Table for a Highway Intersection

|  |  |  |
| --- | --- | --- |
| **Event** | **System State** | **Response** |
| Road sensor detects  vehicle entering left-turn  lane. | Left-turn signal is red. Cross-traffic signal is green. | Start green-to-amber  countdown timer for cross-traffic  signal. |
| Green-to-amber  countdown timer reaches  zero. | Cross-traffic signal is green. | 1. Turn cross-traffic signal amber.  2. Start amber-to-red countdown timer. |
| Amber-to-red  countdown timer reaches  zero. | Cross-traffic signal is amber. | 1. Turn cross-traffic signal red.  2. Wait 1 second.  3. Turn left-turn signal green.  4. Start left-turn-signal countdown timer. |
| Pedestrian presses a  specific walk-request  button. | Pedestrian sign is solid Don’t Walk.  Walk-request countdown timer is not activated. | Start walk-request countdown timer. |
| Pedestrian presses  walk-request button. | Pedestrian sign is solid Don’t Walk.  Walk-request countdown timer is activated. | Do nothing. |
| Walk-request  countdown timer reaches  zero plus the amber  display time. | Traffic signal in walk  direction is green. | Change all green traffic signals to amber. |
| Walk-request  countdown timer reaches  zero. | Traffic signal in walk  direction is amber. | 1. Change all amber traffic signals to red.  2. Wait 1 second.  3. Set pedestrian sign to Walk.  4. Start don’t-walk countdown timer. |

Story Boarding

Following is an example of story boarding which is use to identify and analyze graphically intensive applications.



Fig1: **Home Screen**

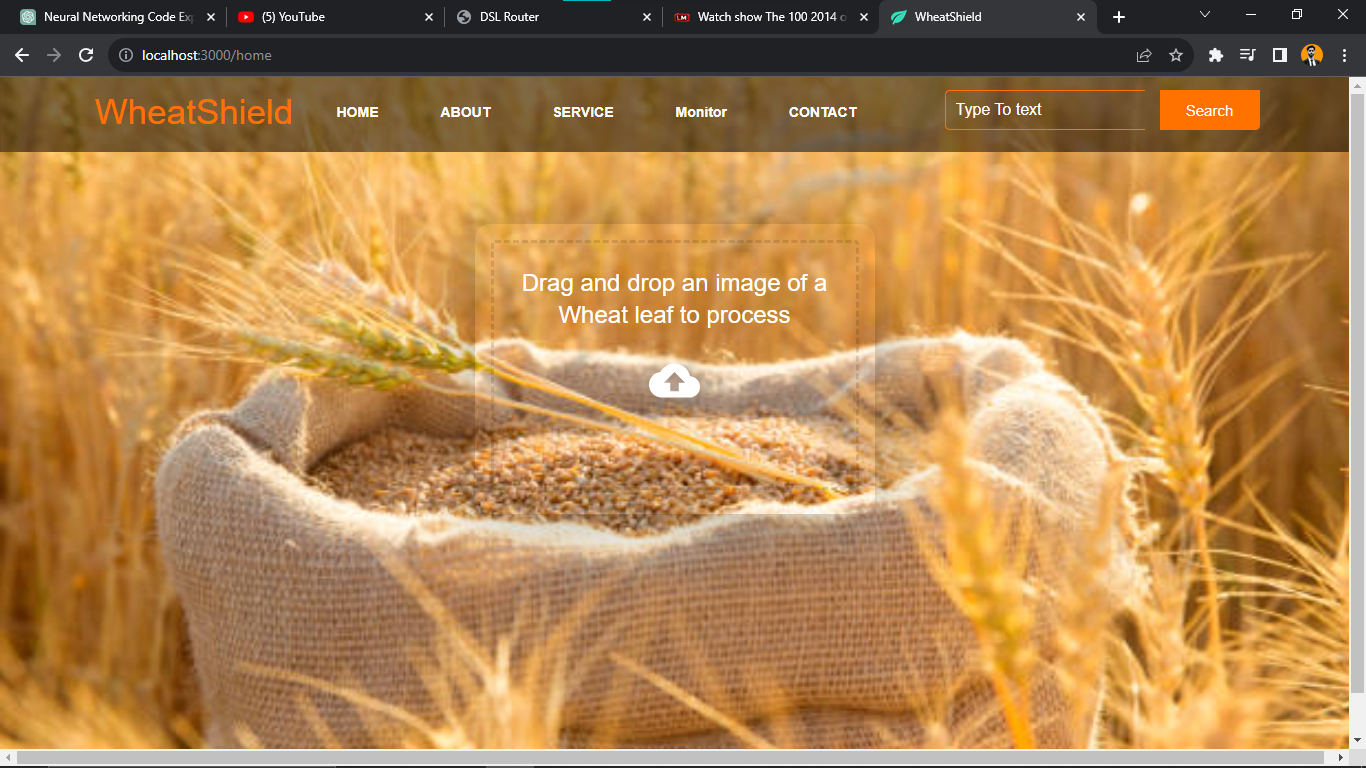


Fig1: **Services Screen (***Disease Detection***)**

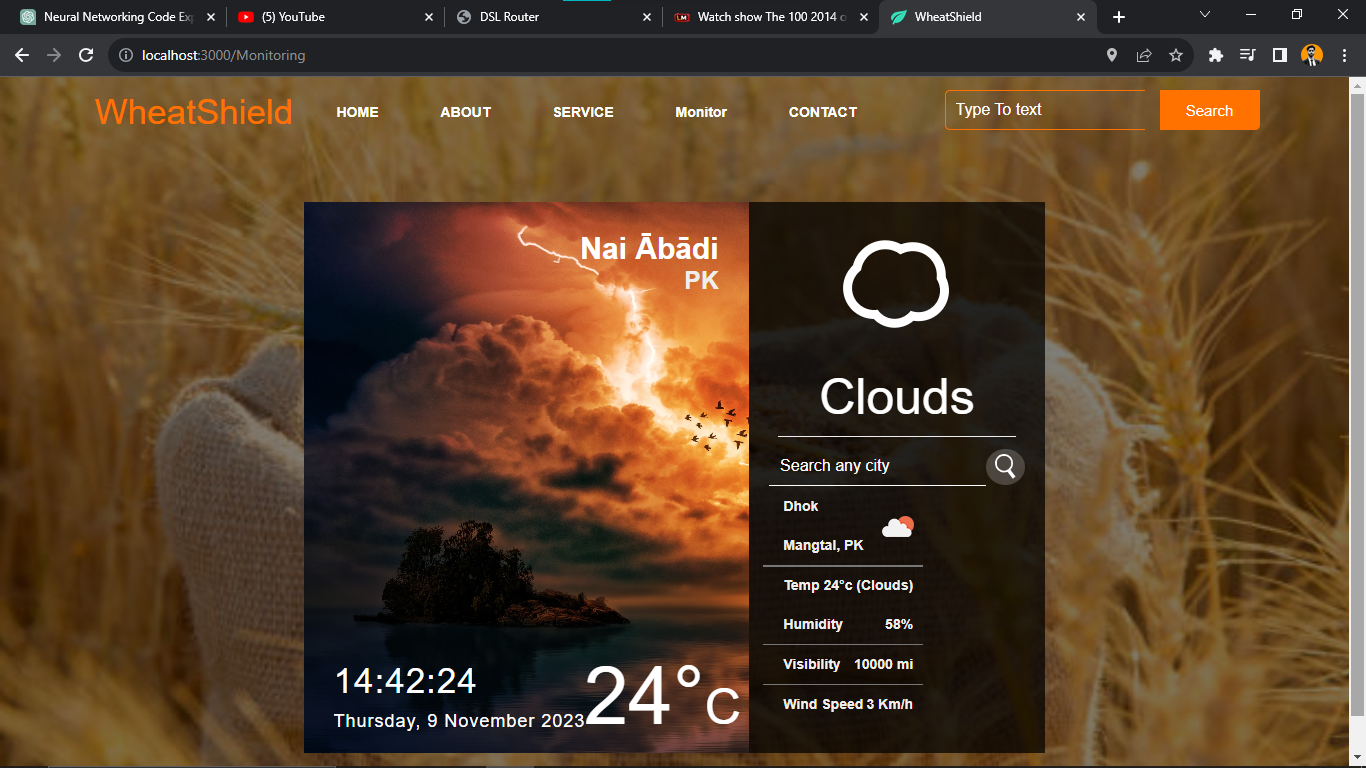


Fig1: **Weather Monitoring Screen**