



**COMSATS University Islamabad (CUI)**

**Software Requirement Specification  
(SRS DOCUMENT)**

**for**

**Title of Project**  
Version 1.0

**By**

**Student Name 1 CIIT/SP18-BCS-xxx/ISB Student Name 2  
CIIT/SP18-BCS-xxx/ISB**

**Supervisor**  
**Supervisor Name**

**Co-Supervisor(if any)**  
**Supervisor Name**

**Bachelor in Computer Science (20xx-20xx)**

**Revision History**

<b>Name</b>	<b>Date</b>	<b>Changes for Reason</b>	<b>Versions</b>
Name	Date	Changes for Reason	Versions
Name	Date	Changes for Reason	Versions
Name	Date	Changes for Reason	Versions

## Application Evaluation History

<b>Comments (by committee) *include the ones given at scope time both in doc and presentation</b>	<b>Action Taken</b>
Name	Date
Name	Date
Name	Date

**Supervisor By**  
**Supervisor's Name**  
Signature \_\_\_\_\_

## Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Purpose . . . . .	5
1.2	Scope . . . . .	5
1.3	Modules . . . . .	5
1.4	Overview . . . . .	5
<b>2</b>	<b>Overall Description</b>	<b>5</b>
2.1	Product Perspective . . . . .	6
2.2	User classes and characteristics . . . . .	6
2.3	Operating Environment . . . . .	6
2.4	Design and Implementation Constraints . . . . .	6
<b>3</b>	<b>Requirement Identifying Technique</b>	<b>7</b>
<b>4</b>	<b>Functional Requirements</b>	<b>7</b>
4.1	Functional Requirement X . . . . .	7
<b>5</b>	<b>Non-Functional Requirements</b>	<b>8</b>
5.1	Reliability . . . . .	8
5.2	Usability . . . . .	9
5.3	Performance . . . . .	9
5.4	Security . . . . .	9
<b>6</b>	<b>External Interface Requirements</b>	<b>9</b>
6.1	User Interfaces Requirements . . . . .	10
6.2	Software interfaces . . . . .	11
6.3	Hardware interfaces . . . . .	11
6.4	Communications interfaces . . . . .	11
<b>7</b>	<b>References</b>	<b>11</b>

## **1 Introduction**

The introduction of the Software Requirements Specification (SRS) should provide an overview of the entire SRS. It should include the purpose, scope, modules and overview of the SRS.

### **1.1 Purpose**

Identify the product or application whose requirements are specified in this document. Describe the different types of reader that the document is intended for, such as developers, project managers, marketing staff, users, testers, and documentation writers.

### **1.2 Scope**

Provide a short description of the software being specified and its purpose. Relate the software to user or project goals and to project objectives. Provide a high-level summary of the major features the software contains or the significant functions that it performs.

### **1.3 Modules**

List the product's major modules, features or user capabilities, emphasizing those that distinguish it from previous or competing products. Give each module or feature a unique and persistent label to permit tracing it to other system elements.

### **1.4 Overview**

This subsection should describe what the rest of the SRS contains and explain how the document is organized.

## **2 Overall Description**

This section presents a high-level overview of the product and the environment in which it will be used, the anticipated users, and known constraints, assumptions, and dependencies.

## 2.1 Product Perspective

Describe the product's context and origin. Is it the next member of a growing product line, the next version of a mature system, a replacement for an existing application, or an entirely new product? Consider including visual models such as a context diagram to show the product's relationship to other systems. See Appendix A for an example of context diagram.

## 2.2 User classes and characteristics

Identify the various user classes that you anticipate will use this product, and describe their pertinent characteristics. See Appendix A for an example of how to present user classes and characteristics.

## 2.3 Operating Environment

Describe the environment in which the software will operate, which might include the hardware platform; operating systems and versions; geographical locations of users, servers, and databases; and organizations that host the related databases, servers, and websites.

Example: **OE-1:** *The System shall operate correctly with the following web browsers: Windows Internet Explorer versions 7, 8, and 9; Firefox versions 12 through 26; Google Chrome (all versions); and Apple Safari versions 4.0 through 8.0.*

## 2.4 Design and Implementation Constraints

There are times when a certain programming language must be used, a code library that has already had time invested to develop it needs to be used, and so forth. Describe any factors that will restrict the options available to the developers and the rationale for each constraint. Constraints are described further in Chapter 14 , “Beyond functionality.” Example:

**CON-1:** *The system shall use the current corporate standard Oracle database engine*

**CON-2:** *The application must use Microsoft .NET framework 4.5.*

**CON-3:** *Online payments may be made only through PayPal.*

**CON-3:** *All textual data used by the application shall be stored in the form of XML files.*

### 3 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 + detail use case)** is an effective technique for interactive end-user applications.
- **Event- response table** is for real-time system in which most of the functionalities are performed at backend.
- **Storyboarding** for graphically intensive applications.

Examples of above techniques are given in Appendix A

### 4 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.

#### 4.1 Functional Requirement X

Itemize the specific functional requirements associated with each feature. These are the software capabilities that must be implemented for the user to carry out the feature’s services or to perform a use case. Describe how the product should respond to anticipated error conditions and to invalid inputs

and actions. Uniquely label each functional requirement, as described earlier. You can create multiple attributes for each functional requirement, such as rationale, source, dependencies, etc. The following template is required to write functional requirements. For further detail see Chapter 11” Writing excellent requirements”.

**Table 1: Description of FR-1**

<b>Identifier</b>	FR-1
<b>Title</b>	Title of requirement
<b>Requirement</b>	Description of requirement which may be written either from the user or system perspective e.g. If written in a user perspective The [user class or actor name] shall be able to [do something] [to some object] [qualifying conditions, response time, or quality statement]. If written in a system perspective [optional precondition] [optional trigger event] the system shall [expected system response]
<b>Source</b>	Where this requirement comes from (who originate it)
<b>Rationale</b>	The motivation behind the requirement
<b>Business Rule (if required)</b>	Any restriction, policy, the rule that the particular requirement must be fulfilled through its functional behavior
<b>Dependencies</b>	Requirements ID that is dependent on this requirement
<b>Priority</b>	High/Medium/Low

## 5 Non-Functional Requirements

This section specifies nonfunctional requirements other than constraints, which are recorded in section 2.3, and external interface requirements, which will appear in section 7. These quality requirements should be specific, quantitative, and verifiable. Chapter 14 “beyond functionality” presents more information about these quality attribute requirements and many examples. The following are some examples of documenting guidelines.

### 5.1 Reliability

Requirements about how often the software fails. The measurement is often expressed in MTBF (mean time between failures). The definition of a failure



must be clear. Also, don't confuse reliability with availability which is quite a different kind of requirement. Be sure to specify the consequences of software failure, how to protect from failure, a strategy for error detection, and a strategy for correction.

## 5.2 Usability

Usability requirements deal with ease of learning, ease of use, error avoidance and recovery, the efficiency of interactions, and accessibility. The usability requirements specified here will help the user interface designer create the optimum user experience. Example:

*USE-1: The COS shall allow a user to retrieve the previous meal ordered with a single interaction.*

## 5.3 Performance

State specific performance requirements for various system operations. If different functional requirements or features have different performance requirements, it's appropriate to specify those performance goals right with the corresponding functional requirements, rather than collecting them in this section. Example:

*PER-1: 95% of webpages generated by the COS shall download completely within 4 seconds from the time the user requests the page over a 20 Mbps or faster Internet connection.*

## 5.4 Security

One or more requirements about protection of your system and its data. The measurement can be expressed in a variety of ways (effort, skill level, time, ...) to break into the system. Do not discuss solutions (e.g. passwords) in a requirements document.

# 6 External Interface Requirements

This section provides information to ensure that the system will communicate properly with users and with external hardware or software elements.

A complex system with multiple subcomponents should create a separate interface specification or system architecture specification. The interface documentation could incorporate material from other documents by reference. For instance, it could point to a hardware device manual that lists the error codes that the device could send to the software.

## 6.1 User Interfaces Requirements

Describe the logical characteristics of each user interface that the system needs. Some possible items to include are

- References to GUI standards or product family style guides that are to be followed.
- Standards for fonts, icons, button labels, images, color schemes, field tabbing sequences, commonly used controls, and the like.
- Screen layout or resolution constraints.
- Standard buttons, functions, or navigation links that will appear on every screen, such as a help button.
- Shortcut keys.
- Message display conventions.
- Layout standards to facilitate software localization.
- Accommodations for visually impaired users.

Document the user interface design details, such as specific dialog box layouts, in a separate user interface specification, not in the SRS. Including screen mock-ups in the SRS to communicate another view of the requirements is helpful but make it clear that the mock-ups are not the committed screen designs. If the SRS is specifying an enhancement to an existing system, it sometimes makes sense to include screen displays exactly as they are to be implemented. The developers are already constrained by the current reality of the existing system, so it's possible to know up front just what the modified, and perhaps the new, screens should look like.

## 6.2 Software interfaces

Describe the connections between this product and other software components (identified by name and version), including other applications, databases, operating systems, tools, libraries, websites, and integrated commercial components (If any). Example:

*SI-1: Cafeteria Inventory System*

*SI-1.1: The COS shall transmit the quantities of food items ordered to the Cafeteria Inventory System through a programmatic interface.*

## 6.3 Hardware interfaces

Describe the characteristics of each interface between the software components and hardware components, if any, of the system. This description might include the supported device types, the data and control interactions between the software and the hardware, and the communication protocols to be used. For more about specifying requirements for systems containing hardware, see Chapter 26, “Embedded and other real-time systems projects.”

## 6.4 Communications interfaces

State the requirements for any communication functions the product will use, including email, web browser, network protocols, and electronic forms.

Example:

*CI-1: The COS shall send an email or text message (based on user account settings) to the Patron to confirm acceptance of an order, price, and delivery instructions.*

## 7 References

List any documents or other resources to which this SRS refers, if any. These might include user interface style guides, standards, system requirements specifications, interface specifications, or the SRS for a related product. The following are a few examples of different resources.

**Book** Author(s). Book title. Location: Publishing company, year, pp. Example: W.K. Chen. Linear Networks and Systems. Belmont, CA: Wadsworth,

1993, pp. 123-35.

**Article in a Journal** Author(s). “Article title”. Journal title, vol., pp, date. Example: G. Pevere. “Infrared Nation.” The International Journal of Infrared Design, vol. 33, pp. 56-99, Jan. 1979.

**Articles from Conference Proceedings (published)** Author(s). “Article title.” Conference proceedings, year, pp. Example: D.B. Payne and H.G. Gunhold. “Digital sundials and broadband technology,” in Proc. IOOC-ECOC, 1986, pp. 557-998.

**World Wide Web** Author(s)\*. “Title.” Internet: complete URL, date updated\* [date accessed]. M. Duncan. “Engineering Concepts on Ice. Internet: [www.iceengg.edu/staff.html](http://www.iceengg.edu/staff.html), Oct. 25, 2000 [Nov. 29, 2003].

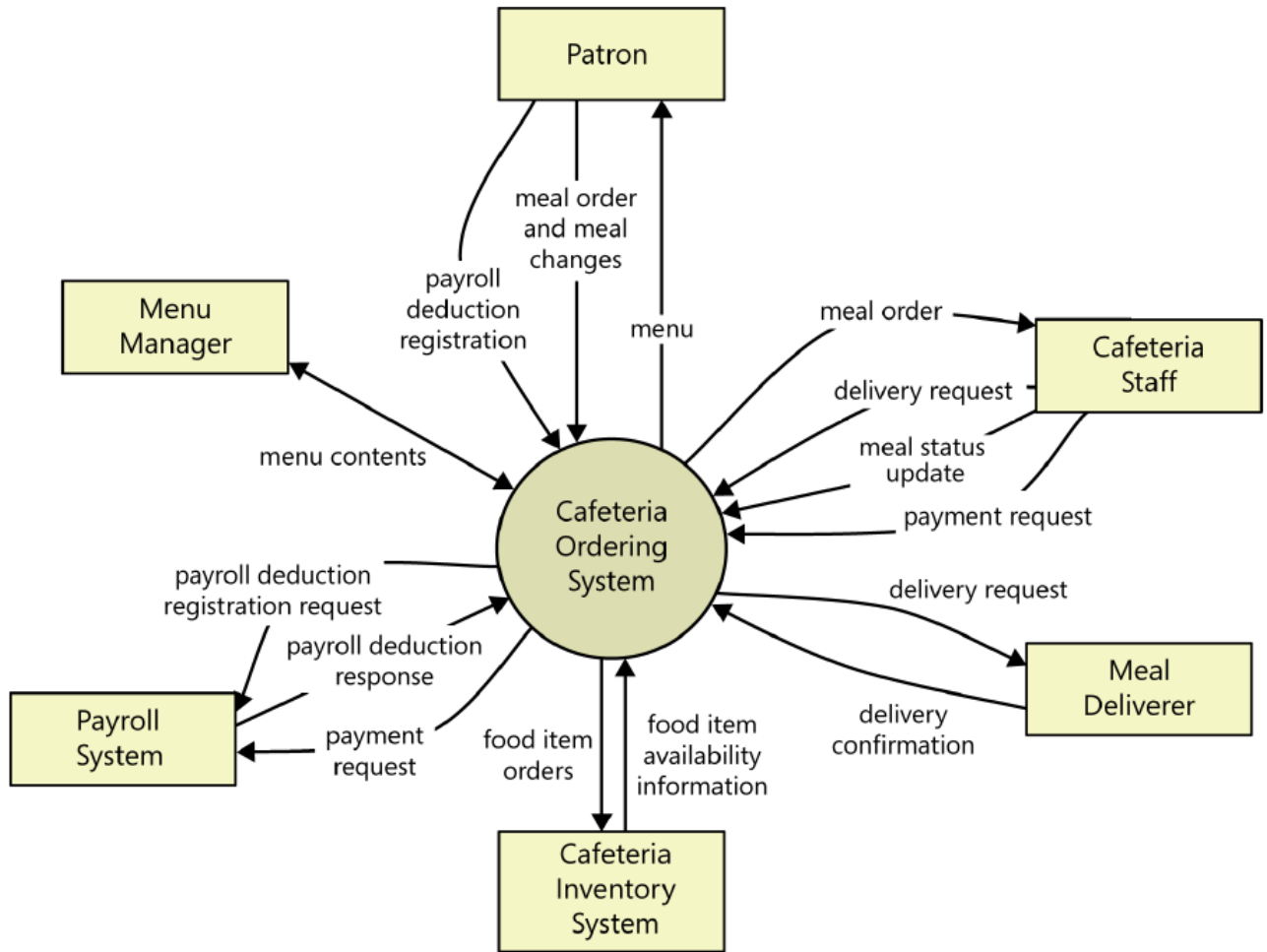


Figure 1: Context diagram of the Cafeteria Ordering System.

## 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 A-1 illustrates the external entities and system interfaces.

## Example: User Classes and Characteristics

**Table A-1 Shows user classes and characteristic for Cafeteria ordering system**

User class	Description
<b>Patron</b>	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.
<b>Cafeteria Staff</b>	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.
<b>Menu Manager</b>	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.
<b>Meal Deliverer</b>	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.

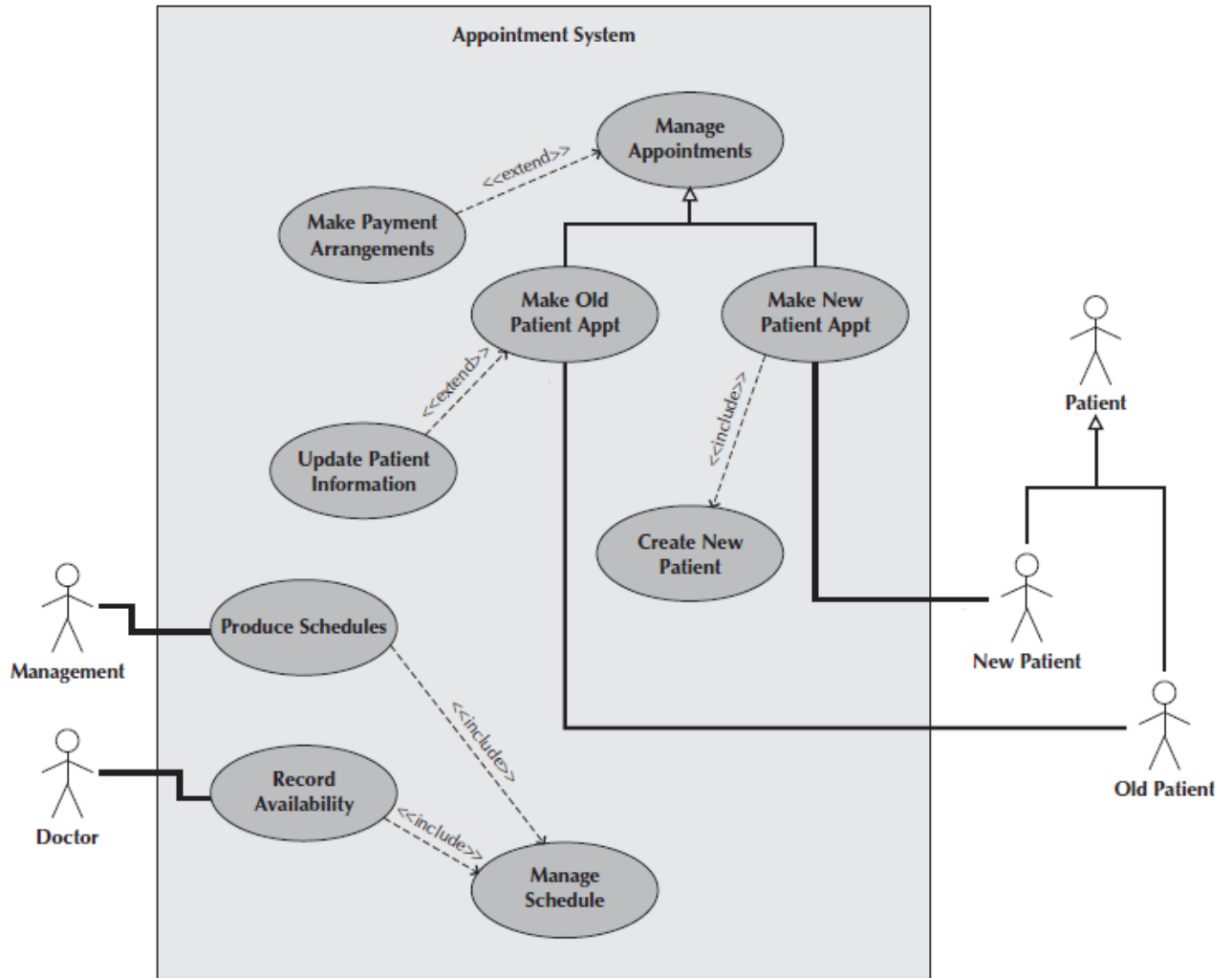






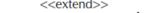



Figure 2: Use Case Diagram of an Appointment System

**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 A-2 to the detail of use case diagram syntax is provided.

# Syntax for Use case Diagram

## Table A-2 Syntax for Use case Diagram

<p><b>An actor:</b></p> <ul style="list-style-type: none"> <li>• Is a person or system that derives benefit from and is external to the subject.</li> <li>• Is depicted as either a stick figure (default) or, if a nonhuman actor is involved, a rectangle with <code>actor</code> in it (alternative).</li> <li>• Is labeled with its role.</li> <li>• Can be associated with other actors using a specialization/superclass association, denoted by an arrow with a hollow arrowhead.</li> <li>• Is placed outside the subject boundary.</li> </ul>	 <p>Actor/Role</p>  <p>&lt;&lt;actor&gt;&gt; Actor/Role</p>
<p><b>A use case:</b></p> <ul style="list-style-type: none"> <li>• Represents a major piece of system functionality.</li> <li>• Can extend another use case.</li> <li>• Can include another use case.</li> <li>• Is placed inside the system boundary.</li> <li>• Is labeled with a descriptive verb-noun phrase.</li> </ul>	 <p>Use Case</p>
<p><b>subject boundary:</b></p> <ul style="list-style-type: none"> <li>• Includes the name of the subject inside or on top.</li> <li>• Represents the scope of the subject, e.g., a system or an individual business process.</li> </ul>	 <p>Subject</p>
<p><b>An association relationship:</b></p> <ul style="list-style-type: none"> <li>• Links an actor with the use case(s) with which it interacts.</li> </ul>	
<p><b>An include relationship:</b></p> <ul style="list-style-type: none"> <li>• Represents the inclusion of the functionality of one use case within another.</li> <li>• Has an arrow drawn from the base use case to the used use case.</li> </ul>	<p>&lt;&lt;include&gt;&gt;</p> 
<p><b>An extend relationship:</b></p> <ul style="list-style-type: none"> <li>• Represents the extension of the use case to include optional behavior.</li> <li>• Has an arrow drawn from the extension use case to the base use case.</li> </ul>	<p>&lt;&lt;extend&gt;&gt;</p> 
<p><b>A generalization relationship:</b></p> <ul style="list-style-type: none"> <li>• Represents a specialized use case to a more generalized one.</li> <li>• Has an arrow drawn from the specialized use case to the base use case.</li> </ul>	



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

**Table A-3 Show the detail use case template and example**

<b>Use Case ID:</b>	Enter a unique numeric identifier for the Use Case. e.g. UC-1
<b>Use Case Name:</b>	Enter a short name for the Use Case using an active verb phrase. e.g. Order a Meal
<b>Actors:</b>	[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
<b>Description:</b>	[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
<b>Trigger:</b>	[Identify the event that initiates the use case.]e.g. A Patron indicates that he wants to order a meal.
<b>Preconditions:</b>	[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.
<b>Postconditions:</b>	[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.
<b>Normal Flow:</b>	[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.
<b>Alternative Flows:</b>	[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.]
<b>Exceptions:</b>	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.
<b>Business Rules</b>	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.
<b>Assumptions:</b>	[List any assumptions. 1. e.g. Assume that 15 percent of Patrons will order the daily special (Source: previous 6 months of cafeteria data).

aslkfjakldjfa;kl