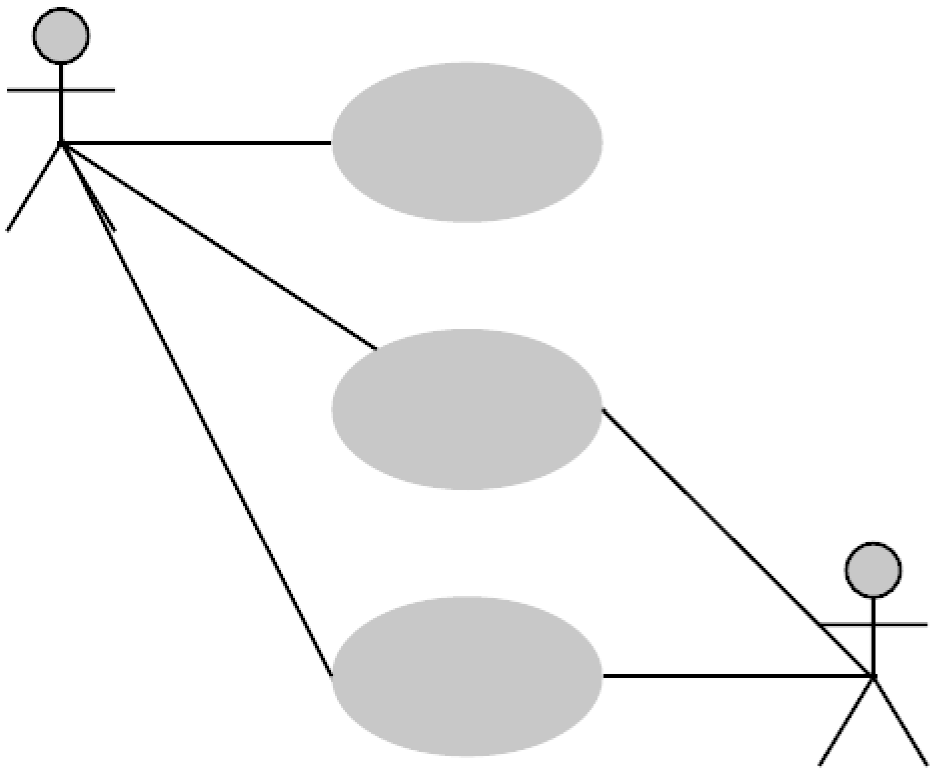
UML Use Case Diagram

The following sections will show what are use case diagrams, when should we apply use case diagrams, the components which form the use case diagram and a small example.

What is a use case diagram?

In the Unified Modeling Language (UML), a use case diagram can summarize the details of your system's users (also known as actors) and their interactions with the system. To build one, you'll use a set of specialized symbols and connectors. An effective use case diagram can help your team discuss and represent:

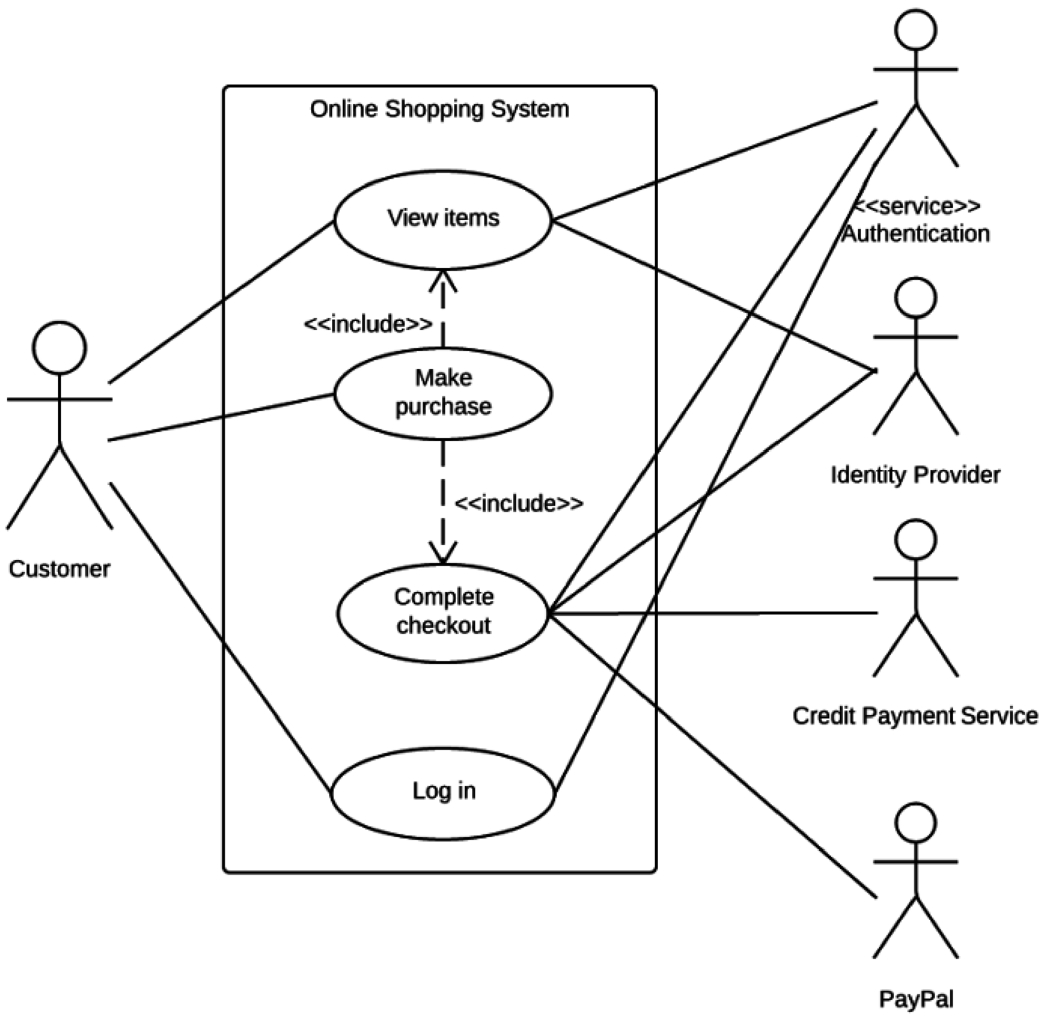
- Scenarios in which your system or application interacts with people, organizations, or external systems
- Goals that your system or application helps those entities (known as actors) achieve
- The scope of your system



When to apply use case diagrams

A use case diagram doesn't go into a lot of detail—for example, don't expect it to model the order in which steps are performed. Instead, a proper use case diagram depicts a high-level overview of the relationship between use cases, actors, and systems. Experts recommend that use case diagrams be used to supplement a more descriptive textual use case.
UML is the modeling toolkit that you can use to build your diagrams. Use cases are represented with a labeled oval shape. Stick figures represent actors in the process, and the actor's participation in the system is modeled with a line between the actor and use case. To depict the system boundary, draw a box around the use case itself.
UML use case diagrams are ideal for:

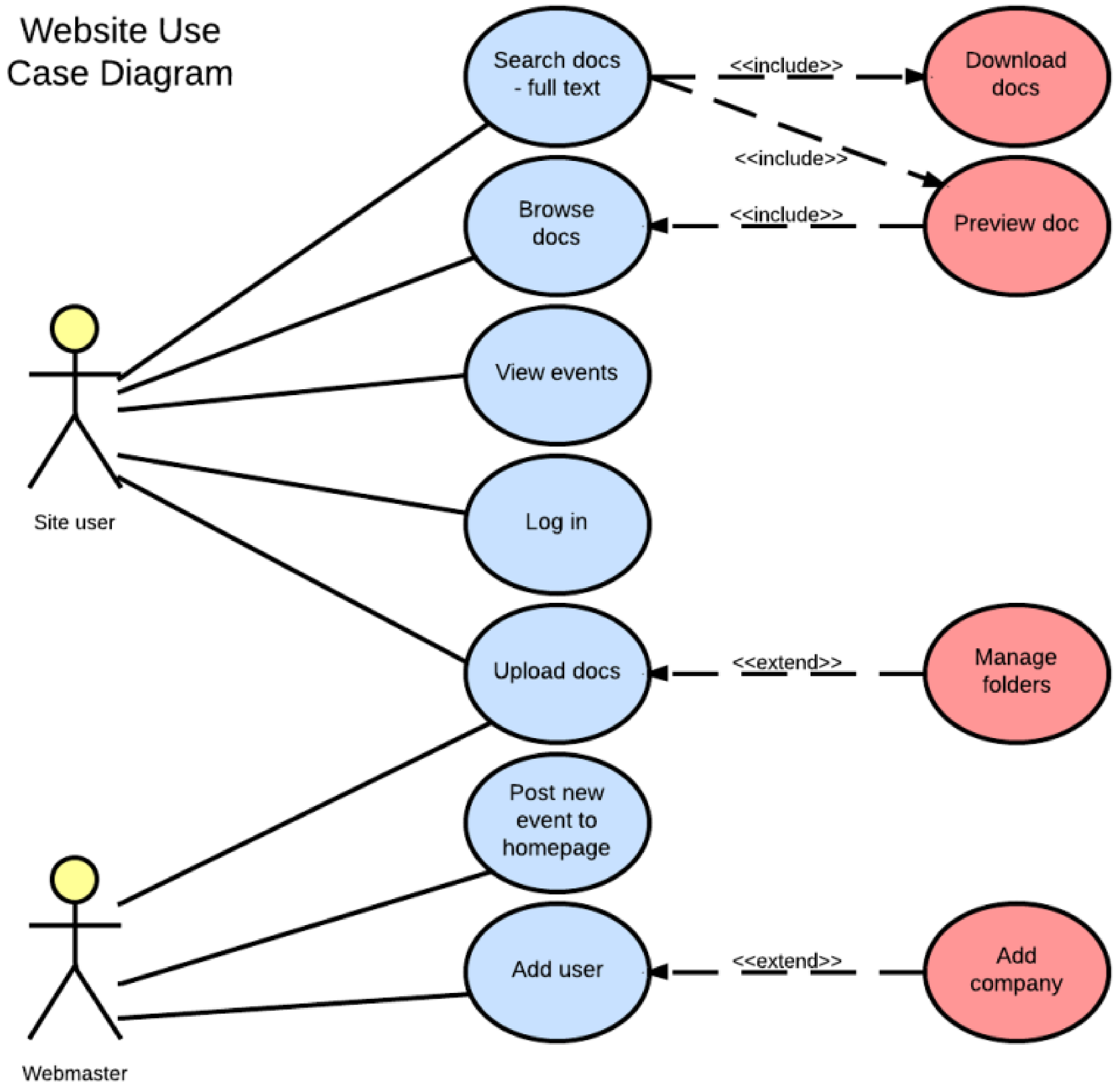
- Representing the goals of system-user interactions
- Defining and organizing functional requirements in a system
- Specifying the context and requirements of a system
- Modeling the basic flow of events in a use case



Use case diagram components

To answer the question, "What is a use case diagram?" you need to first understand its building blocks. Common components include:

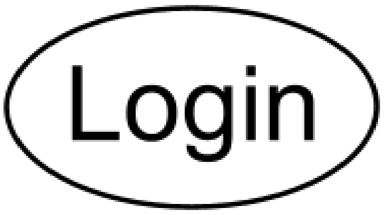
- **Actors:** The users that interact with a system. An actor can be a person, an organization, or an outside system that interacts with your application or system. They must be external objects that produce or consume data.
- **System:** A specific sequence of actions and interactions between actors and the system. A system may also be referred to as a scenario.
- **Goals:** The end result of most use cases. A successful diagram should describe the activities and variants used to reach the goal.



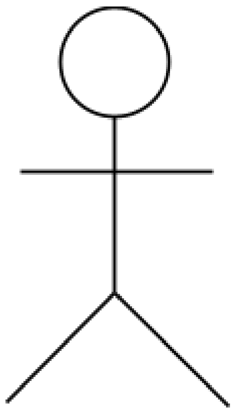
Use case diagram symbols and notation

The notation for a use case diagram is pretty straightforward and doesn't involve as many types of symbols as other UML diagrams:

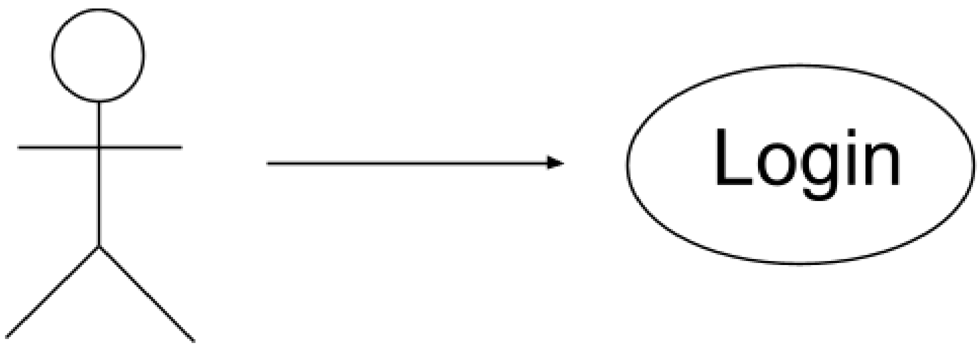
- **Use cases:** Horizontally shaped ovals that represent the different uses that a user might have.



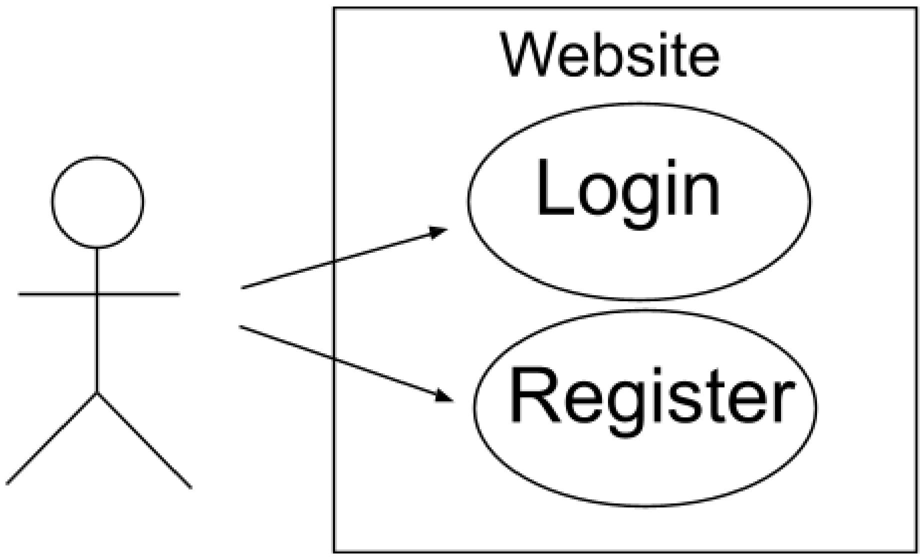
- **Actors:** Stick figures that represent the people actually employing the use cases.



- **Associations:** A line between actors and use cases. In complex diagrams, it is important to know which actors are associated with which use cases.

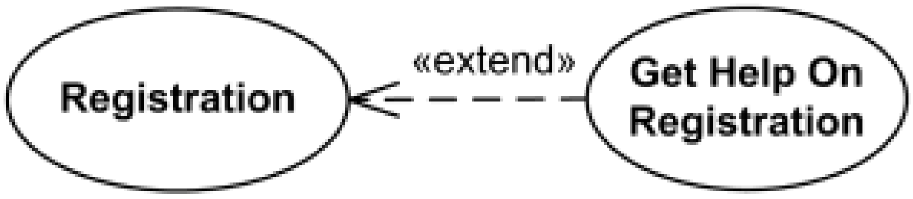


- **System boundary boxes:** A box that sets a system scope to use cases. All use cases outside the box would be considered outside the scope of that system. For example, Psycho Killer is outside the scope of occupations in the chainsaw example found below.



Between the use cases there can be define two types of relations: **extend** and **include.**

Extend is a directed relationship that specifies how and when the behavior defined in usually supplementary (optional) **extending use case** can be inserted into the behavior defined in the **extended use case**.
Extended use case is meaningful on its own, it is **independent** of the extending use case. **Extending** use case typically defines **optional** behavior that is not necessarily meaningful by itself. The extend relationship is **owned** by the extending use case. The same extending use case can extend more than one use case, and extending use case may itself be extended.
Extend relationship is shown as a dashed line with an open arrowhead directed from the **extending use case** to the **extended (base) use case**. The arrow is labeled with the keyword **«extend»**.

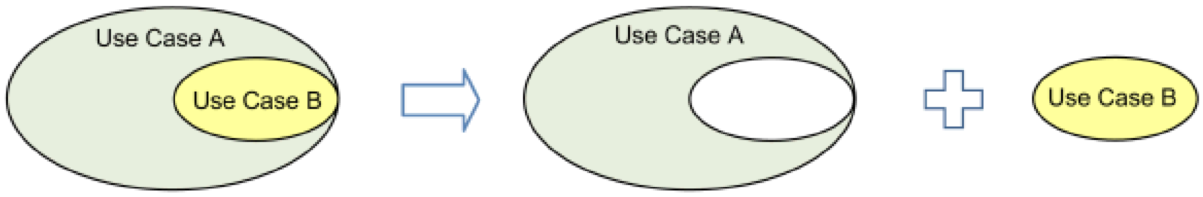


Registration use case is complete and meaningful on its own.
It could be extended with optional **Get Help On Registration** use case.

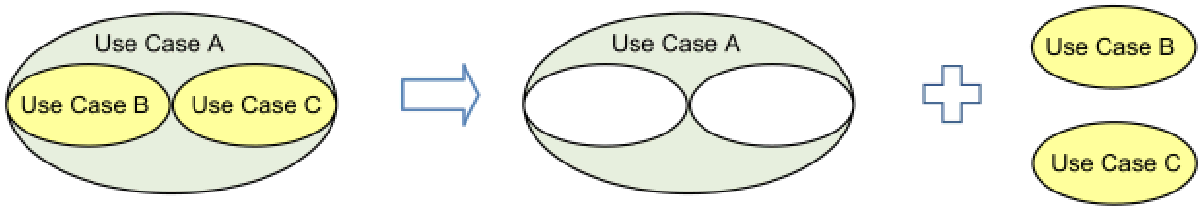
Use case include is a directed relationship between two use cases which is used to show that behavior of the **included** use case (the addition) is inserted into the behavior of the **including** (the base) use case.
The **include** relationship could be used:

- to simplify large use case by splitting it into several use cases,
- to extract **common parts** of the behaviors of two or more use cases.

A large use case could have some behaviors which might be detached into distinct smaller use cases to be included back into the base use case using the UML **include** relationship. The purpose of this action is modularization of behaviors, making them more manageable.

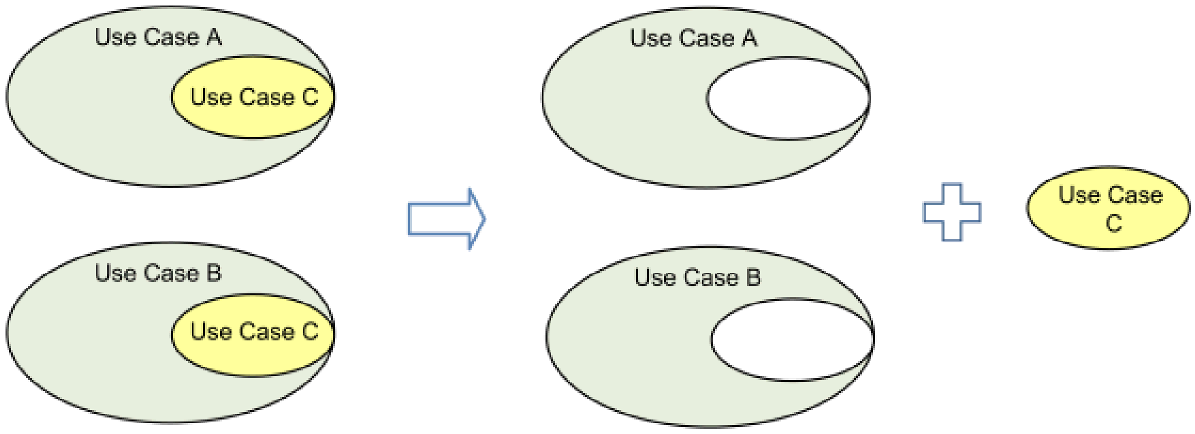


Use case B is extracted from larger use case A into a separate use case.



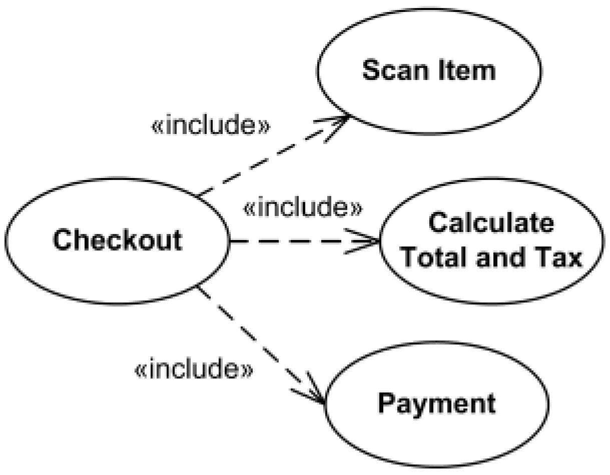
Use cases B and C are extracted from larger use case A into separate use cases.

When two or more use cases have some common behavior, this common part could be extracted into a separate use case to be included back by the use cases with the UML **include** relationship.



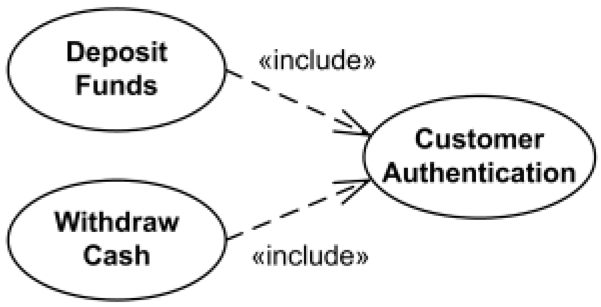
Use case C is extracted from use cases A and B to be reused by both use cases using UML include relationship.

Include relationship between use cases is shown by a dashed arrow with an open arrowhead from the including (base) use case to the included (common part) use case. The arrow is labeled with the keyword «include».



Checkout use case includes several use cases - Scan Item, Calculate Total and Tax, and Payment

Large and complex Checkout use case has several use cases extracted, each smaller use case describing some logical unit of behavior. Note, that including Checkout use case becomes incomplete by itself and requires included use cases to be complete.

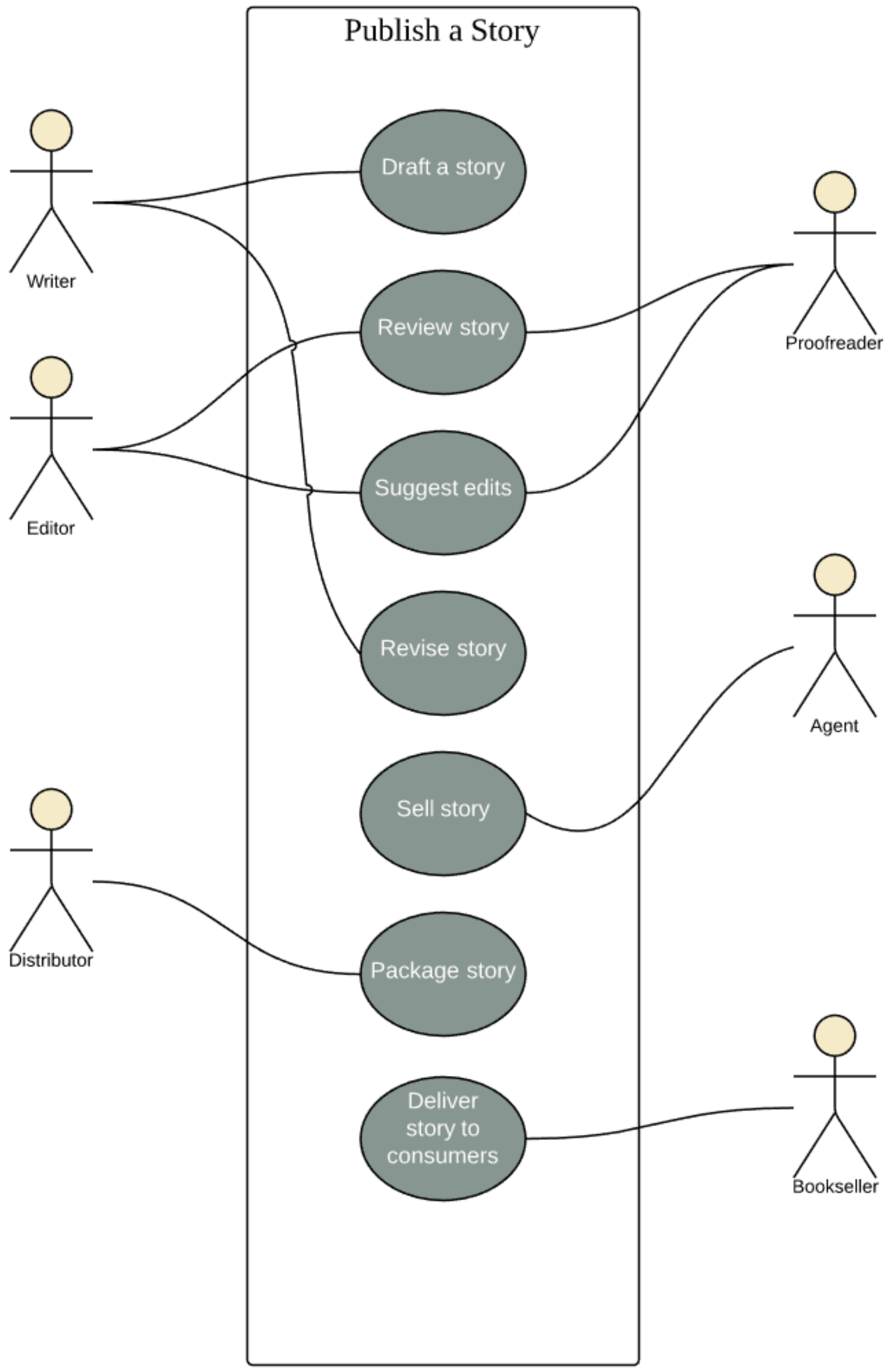


Deposit Funds and **Withdraw Cash** use cases include **Customer Authentication** use case.

Use case diagram examples

Book publishing use case diagram example

This use case diagram is a visual representation of the process required to write and publish a book. Whether you’re an author, an agent, or a bookseller, inserting this diagram into your use case scenario can help your team publish the next big hit.



The cake-shop use case diagram example

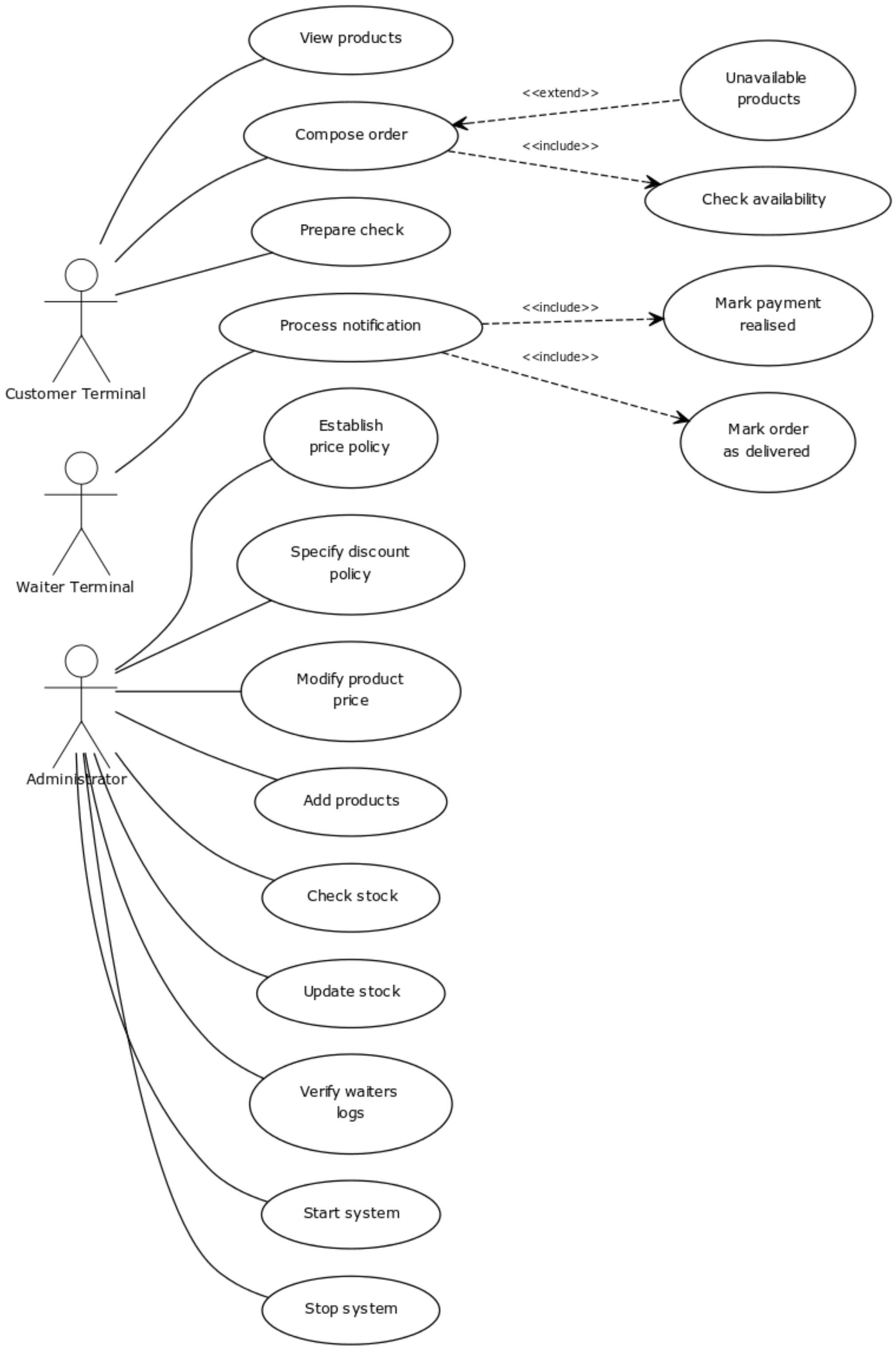
We have the following document containing the description of an system we have to build.

An eccentric cake-shop owner wants to introduce a software system to support the basic activities in her shop. On each table, there is a terminal connected to a central computer. A customer (all the people at a table) can use the terminal to consult the products sold in the cake-shop (e.g., refreshments of different types, various kinds of cakes, various types of alcoholic drinks, etc.) each of which having a description and a price. If the customer wants to, she can compose an order specifying all the products and the associated quantities. After finishing the order, the system checks the availability of each product (in the desired quantity). If some products are not available in the desired quantity, those products are eliminated from the order and the system offers to the customer a list of alternative products to select from. When all the products in the order are available, the systems updates the stock database, and the order is sent to a waiter. The waiter is selected by the system in such a way that all the waiters will get a similar number of orders in a working day. Additionally, all the orders from the same customer will be sent to the same waiter (after a time, a customer can sent another order). When a customer decides to leave the shop, the customer sends a notification to the corresponding waiter to prepare the check.

Each waiter has a personal terminal. Using it, the system notifies the waiter that she received an order from a particular table. All the products are specified together with the desired quantities for each product. In a similar way, the system notifies a waiter that she must go to a particular table to give and cash the customer check. The system will also compute and provide the total value of all the orders of that customer. When a waiter receives a notification (of any kind) the system records it in a log specific to that waiter. The waiter processes each notification in the order in which they are received. In the case of an order notification, the waiter prepares the order content (i.e., puts all the products on a plateau) and delivers it to the corresponding table. After that, he marks the order as being delivered and the system logs this fact and the delivery time in the waiter log. In the case of a check notification, the waiter goes to the table and cash the money. She marks using the terminal that the payment is performed and then the system updates the waiter total amount of money she received and prints the customer receipt containing all the products, prices, quantities and the total value of the receipt. The moment of payment is logged in the waiter log.

From the waiter point of view, computing the total value of a check is very important. This value is obviously based on the price of each product and on the consumed quantity. However, each day, there are some special products for which a discount is given. A special product is a combination of at least two normal products (e.g., a coke and a cheese-cake). The price of a special product is computed as the sum of the prices of each component product from which we subtract a percent specific to that special product. Moreover, an additional discount can be given based on the values of all the orders of a customer: in one day no discount may be given; in another day, if the total value is greater than a specified minimum value a fixed discount is given e.g., 20 RON; in another day, if the total value is greater than a specified minimum value a percent discount is given e.g., 10%; in another day, the discount can be combined (fixed amount or percent) and the discount value is computed in such a way in which to favor the client or the company (depending on how the system is configured).

The price policy is established by an administrator that can compose special products to be available in the next working day. Additionally, the administrator can specify what discount policy is going to be applied the next day. Moreover, she can modify the price of a product starting with the next day, can add products to be available starting with the next day, can check the stock in order to ask producers to supply the shop with various products. The stock can also be updated by the administrator when new products arrive in the shop and the verification of the waiters logs can also be performed by the administrator. At the beginning of a working day, the administrator starts the system. At the end of a working day, the administrator stops the system. In that case, a report is produced by the system containing the total income for that day and the total income of each waiter.


View products
Compose order
Prepare check
Process notification
Unavailable products
Check availability
Mark payment realised
Mark order as delivered
<<extend>>
<<include>>
<<include>>
<<include>>
Customer Terminal
Waiter Terminal
Administrator
Establish price policy
Specify discount policy
Modify product price
Add products
Check stock
Update stock
Verify waiters logs
Start system
Stop system