**TITLE:**

**“COOK BOOK XPRESS”**



**SDAQ0511**

**A Team of**

**Nallala Mohana Lakshmi -21B91A05L0 (CAN\_25092955)**

**Pithani Anitha -21B91A05P7 (CAN\_25093384)**

**Mandava Sushma Chowdary -21B91A05I4 (CAN\_25092964)**

**Trained by**

**Dr. K V Krishnam Raju**

1. **Abstract**

**Title**: Enhancing Culinary Experiences Through Cooking Vlogs and Instant Ingredient Delivery.

In today's fast-paced world, people are increasingly turning to online platforms for culinary inspiration and convenience. This project aims to bridge the gap between culinary enthusiasts and their culinary aspirations by combining the power of cooking vlogs and instant ingredient delivery. The project follows a structured Software Development Life Cycle (SDLC) process. Our innovative solution involves the creation of a user-friendly mobile application that connects users with a diverse array of cooking vlogs, hosted by talented chefs and home cooks. Users can browse through a vast library of video recipes, each accompanied by detailed ingredient lists and step-by-step instructions. What sets our project apart is the integration of an instant ingredient delivery system. With a single tap, users can add the required ingredients to their shopping cart and have them delivered to their doorstep within minutes. This seamless integration not only simplifies the cooking process but also ensures that users have access to fresh and high-quality ingredients, enhancing the overall cooking experience. Throughout the project's development, we adhere to the SDLC framework, emphasizing requirements analysis, design, implementation, testing, and maintenance.

In summary, "Enhancing Culinary Experiences Through Cooking Vlogs and Instant Ingredient Delivery" leverages the power of digital media and instant delivery to transform cooking into a convenient and delightful journey.

1. **Software Requirement Specification**
   1. **Introduction**

* An idea of introducing a new era of webstore with food blogging which helps the end users (consumers).
  + 1. **Purpose**
* An application which brings together the joy of cooking with convenience of instant ingredient delivery.
* To build up the muscle of E-grocery and to foster the cause of culinary Hub.
  + 1. **Scope**
* The scope encompasses the creation of an e-commerce website where users can explore a wide range of cuisine vlogs with the facility of picking up the grocery on-spot with an instantaneous secured shipment.
  1. **Existing System**
* The ongoing trend of the food delivery organizations is that it delivers the conceivable food within no timespan from our desired restaurant.
* As well there are many platforms for food vlogs where one can follow, pick up and make effort to cook on their own.
  + 1. Food Delivery Organizations:
* There are many food delivery organizations like Swiggy, Zomato etc., who delivers the delicious cooked and baked food.
* They made customers feel at ease by providing a facility of quick home delivery.
* This helped a lot for the current generation.
  + 1. Platforms for vlogs:
* As well there are many platforms where an individual can watch videos of preparation of a multi-cuisine at home.
* By watching these videos users are enthusiastic to learn and to cook the food on their own.
  1. **Proposed System**
     1. Overview
* This website contains mainly two modules of which one is that it provides videos of various dishes and their cooking methods.
* And another one is that it facilitates the customers by delivering the components of the dish if they are in need.
* The platform aims to inspire and assist users in making delicious meals at home by providing the user-friendly interface with an extra feature of ordering the necessities.
  + 1. Functional Requirements

2.3.2.1 FR1

* Title: Access the website
* Desc: Open the website and go through the videos in it.

2.3.2.2 FR2

* Title: Choose video of your interest
* Desc: Now choose the video that you wish to watch or the dish you would like to cook.

2.3.2.3 FR3

* Title: Ingredients in description.
* Desc: The ingredients used in the recipe are displayed in the description.

2.3.3.4 FR4

* Title: Prominent search feature
* Desc: The search bar should be prominent and easy to find.

2.3.3.5 FR5

* Title: Usage of search bar
* Desc: If the user wants a specific recipe then they can approach search bar.

2.3.3.6 FR6

* Title: Sign up or login
* Desc: If the user has an account then he/she can directly login to the website if not he/she should sign up.

2.3.2.7 FR7

* Title: Check the ingredients
* Desc: Select the items that you are in need of.

2.3.2.8 FR8

* Title: Add to cart
* Desc: Add the selected items to cart by clicking on add to cart button.

2.3.2.9 FR9

* Title: Cart Button
* Desc: To order the items, go to cart and then select the set of items you need.

2.3.2.10 FR10

* Title: Buy now
* Desc: After selecting the set of items click on buy now button.

2.3.2.11 FR11

* Title: Fill in the address
* Desc: Before you place the order provide the delivery address where you want your order to be delivered.

2.3.2.12 FR12

* Title: Choose the payment method
* Desc: You should choose the method you wanted to pay for example:

1. Cash On Delivery
2. UPI payment
3. Debit Card

2.3.2.13 FR13

* Title: Place the order
* Desc: After providing every detail place the order.

2.3.2.14 FR14

* Title: Notify the customer
* Desc: After placing the order the customer will be notified with details of the delivery and intimated with time of arrival of the items.

2.3.2.15 FR15

* Title: Track the order
* Desc: Provide a GPS tracker for the customer so that they can track the order.
  + 1. Non-Functional Requirements

2.3.3.1 Performance

1. PR1

* Title: Response Time
* Desc: The fastness of the search and response time after every click.
* Must: No more than 2 seconds 100% of the time.
* Wish: No more than 1 second 100% of the time.

1. PR2

* Title: System Dependability
* Desc: The fault tolerance of the system. If the system loses the connection to the Internet or to the GPS device or the system gets some strange input, the user should be informed.
* Must: 100% of the time.

1. PR3

* Title: Database Performance
* Desc: Database queries shall be optimized to ensure fast and efficient retrieval of data.
* Must: No more than 5 seconds 100% of the time.
* Wish: No more than 2 second 100% of the time.
  + - 1. Availability

1.AB1

* Title: System Availability
* Desc: The availability of the system when it is used with no 404 error.
* Must: More than 98% of the time.
* Wish: 100% of the time.

2.AB2

* Title: Internet Connection
* Desc: The application should be connected to the Internet in order to communicate with the database.
* Must:100% of the time of usage.

3.AB3

* Title: GPS Connection
* Desc: The application should be connected to the GPS in order to get the users location, the map and to calculate the distance
* Must:100% of the time of usage.

4.AB4

* Title: Search results Availability
* Desc: The availability of the search results should be maximum so that every user will be satisfied
* Must: More than 90% of the time.
* Wish:100% of the time.
  + - 1. Reliability

1.RB1

* Title: System Reliability for search results
* Desc: The system gives the right result on a search.
* Must: More than 98% of the searches.
* Wish:100% of the searches.

2.RB2

* Title: Reliability for ingredients.
* Desc: The application provides the correct ingredients used in the recipe.
* Must: More than 98% of the time.
* Wish:100% of the time.
  + - 1. Security

1. SR1

* Title: User data Protection
* Desc: The data provided by the user will be protected and kept private from the external user or attacker and will be encrypted.
* Must: More than 99.9999% of the time.
* Wish: 100% of the time.

1. SR2

* Title: Authentication and Authorization
* Desc: The system will use secure authentication. Passwords will be stored in hashed and salted formats to prevent unauthorized access.
* Must: More than 99.99999999% of the time.
* Wish: 100% of the time.

1. SR3

* Title: Security of accounts
* Desc: If the user provides wrong username or password then the system should notify them and they shouldn’t be logged in.
* Must: More than 99.99999999% of the time.
* Wish: 100% of the time.
  + - 1. Maintainability

1. MB1

* Title: Application extendibility
* Desc: The application should be easy to extend. The code should be written in a way that it favors implementation of new functions.
* Span: 365 days 24x7.

1. MB2

* Title: Application Testability
* Desc: Test environments should be built for the application to allow testing of the application different functions.
* Accuracy: 100% of whole.
  1. **Pseudo Requirements**
     1. Software Requirements
        1. Programming Languages

Front-end

1. HTML
2. CSS
3. Java script
4. React.js

Back-end

1. PHP
2. Node.js
3. SQL
   * + 1. Database Management System
4. Oracle
   * 1. Hardware Requirements
        1. Operating System
     2. Windows
        1. Processors and Storage
5. RAM : 4 GB
6. ROM : 50 GB
7. Hard Disk : 512 TB
8. Processor : i5
   1. **Use Case Description**
9. **Use Case Name**: "Order Ingredients from Food Videos".
10. **Customer/User**: The primary actor in an online shopping application. Users browse products, add items to their cart, place orders, and interact with various features of the application.

**Guest User**: A user who browses the website without logging in. They might have limited access to certain features, and their browsing behavior can be tracked to encourage them to create an account.

**Customer Support**: Representatives who provide assistance to users regarding their orders, account issues, returns, and other inquiries.

1. **Description:** This use case describes the process by which users can watch food videos showcasing different dishes and, upon finding a dish they'd like to cook, have the option to instantly order the ingredients needed for that dish.
2. **Preconditions:** 
   * + - 1. User is logged into the application.
         2. User is watching a food video.
3. **Postconditions**:
   * + - 1. User's selected ingredients are added to their shopping cart.
         2. User has the option to proceed to checkout.
4. **Main Flow of Events:**
   * + - 1. User browses and selects a food video to watch.
         2. While watching the video, the user can view a list of ingredients required for the dish being prepared.
         3. The user decides to order the ingredients and selects the "Order Ingredients" option. The application adds the selected ingredients to the user's shopping cart.
         4. The user has the option to continue watching videos or proceed to checkout.
5. **Alternative Flows:** If the user decides not to order ingredients, they can continue watching videos without adding items to the cart.
6. **Trigger:** User selects the "Order Ingredients" option while watching a food video.
7. **Dependencies**: Relies on the availability of accurate ingredient information.
8. **UML Diagram**

Unified Modeling Language (UML) is a standardized visual language used in software engineering to model and design software systems. It provides a set of diagrams and notation conventions that help communicate various aspects of a system's architecture, behavior, interactions, and structure. Here are some of the most commonly used UML diagrams:

* 1. **Use Case Description:** Represents the interactions between actors and the system, capturing the functional requirements of the system from a user's perspective.
  2. **Class Diagram**: Depicts the structure of the system by illustrating classes, their attributes, methods, relationships, and inheritance hierarchies.
  3. **Sequence Diagram**: Shows the interactions and message exchange between different objects or components over a period of time, representing dynamic behavior.
  4. **Activity Diagram:** Visualizes the flow of activities and actions within a process or workflow, showing the sequence of steps and decision points.
  5. **State Chart Diagram**: Models the behavior of an individual object or component by capturing the different states it can be in and the transitions between those states.
  6. **Collaboration** **Diagram**: Similar to sequence diagrams, it focuses on the structural organization of objects and their interactions.
  7. **Use Case Description**

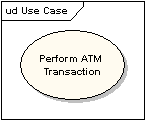
### Actors

### A use case diagram shows the interaction between the system and entities external to the system. These external entities are referred to as actors. Actors represent roles which may include human users, external hardware or other systems. An actor is usually drawn as a named stick figure, or alternatively as a class rectangle with the «actor» keyword. Eg:

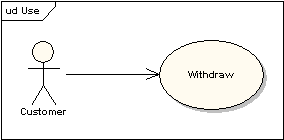
|  |  |
| --- | --- |
| UML Actor1 | UML Actor2 |

### Use Cases

### A use case is a single unit of meaningful work. It provides a high-level view of behaviour observable to someone or something outside the system. The notation for a use case is an ellipse. Eg:

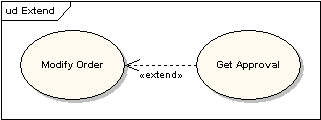


1. Arrow  
   The notation for using a use case is a connecting line with an optional arrowhead showing the direction of control. The following diagram indicates that the actor "Customer" uses the "Withdraw" use case. Eg:

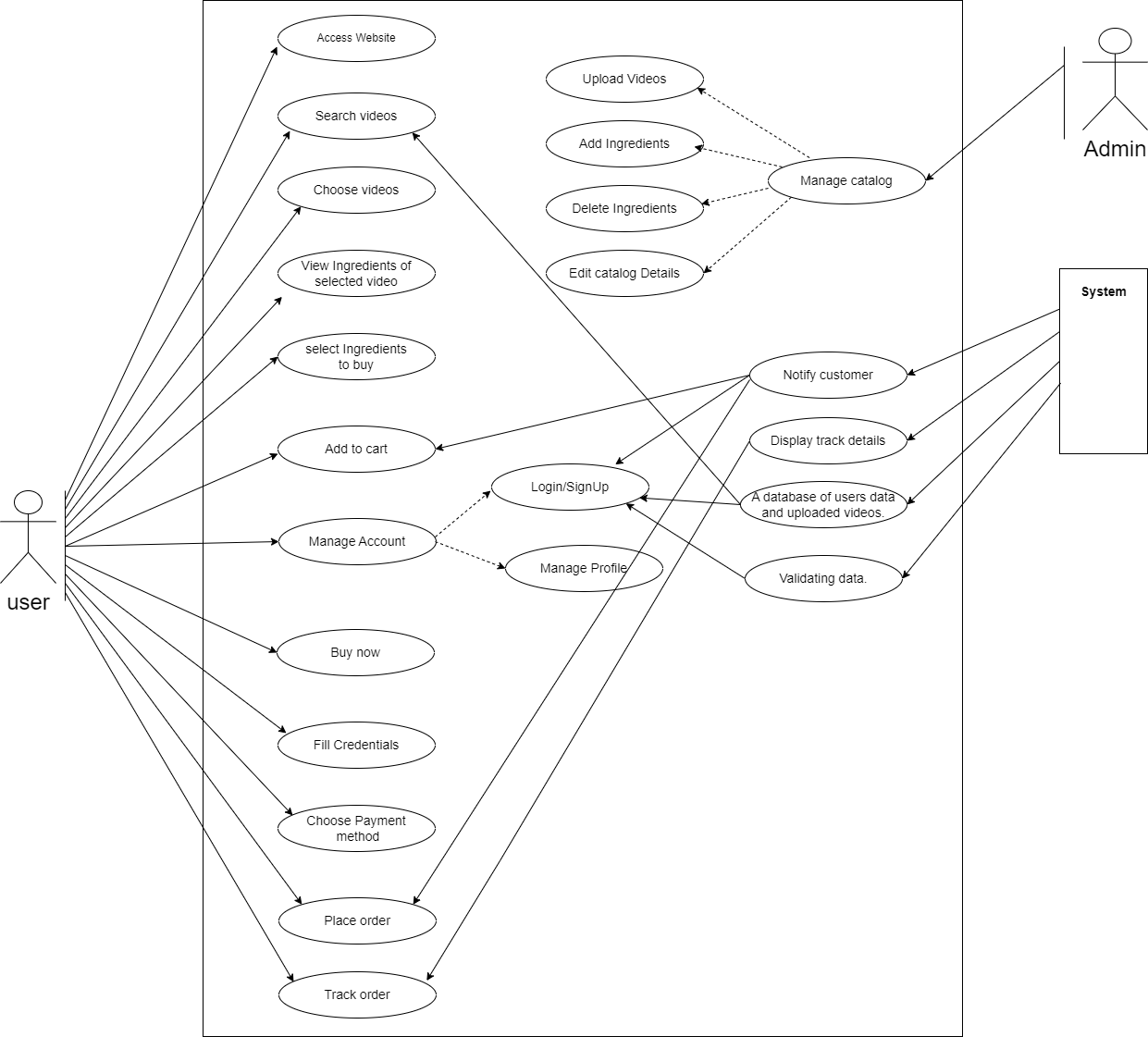


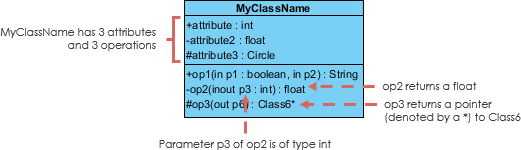
### Extending Use Cases

One use case may be used to extend the behavior of another; this is typically used in exceptional circumstances. Eg:



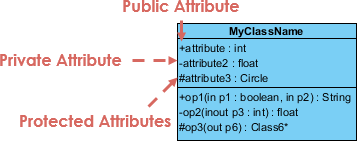
Use case Representation



* 1. **Class Diagram**
     + - 1. Class Name: The name of the class appears in the first partition.
         2. Class Attributes:
* Attributes are shown in the second partition.
* The attribute type is shown after the colon.
  + - * 1. Class Operations (Methods):
* Operations are shown in the third partition. They are services the class provides. The return type of a method is shown after the colon at the end of the method signature. 

### **Class Visibility**

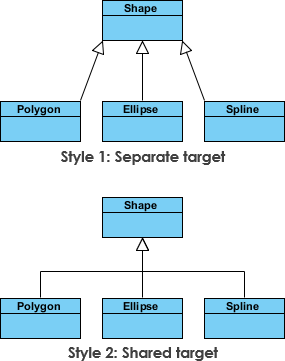
### The +, - and # symbols before an attribute and operation name in a class denote the visibility of the attribute and operation.

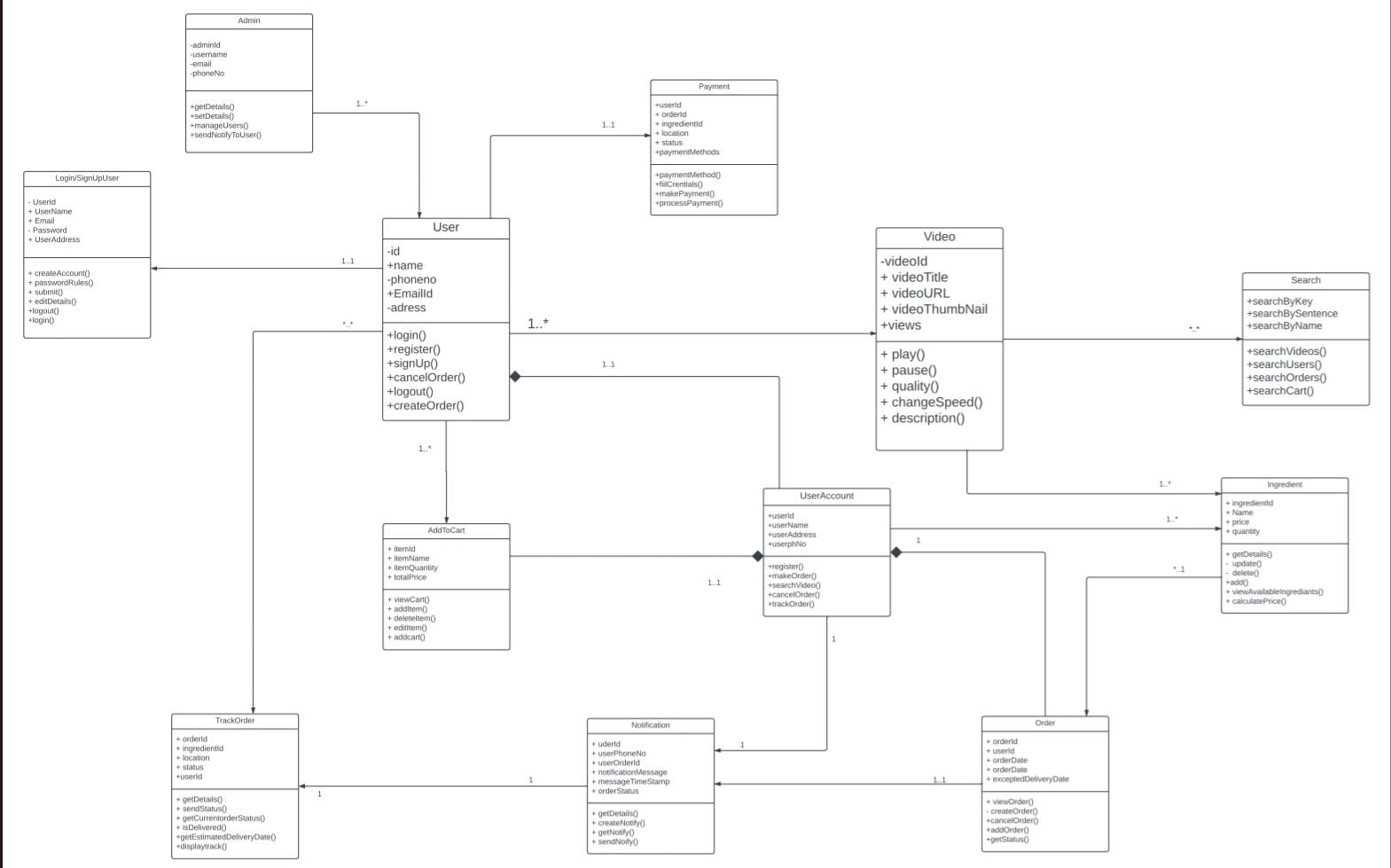


* + denotes public attributes or operations
* - denotes private attributes or operations
* # denotes protected attributes or operations

### **Inheritance Example - Shapes**

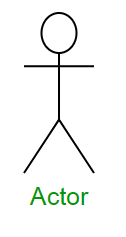
### The figure below shows an inheritance example with two styles. Although the connectors are drawn differently, they are semantically equivalent.



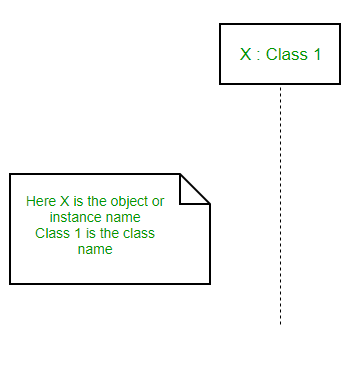


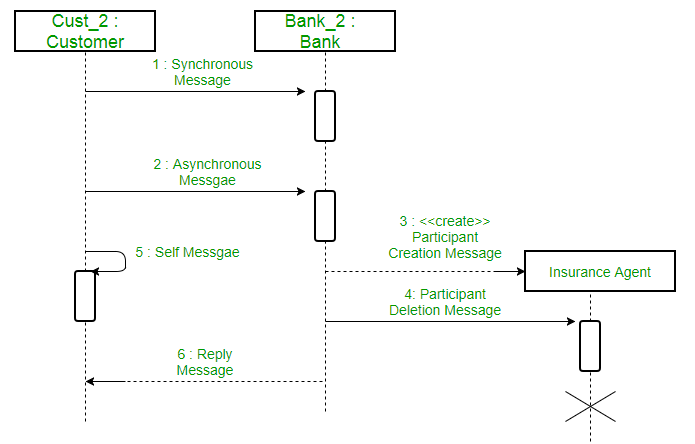
* 1. **Sequence Diagram**

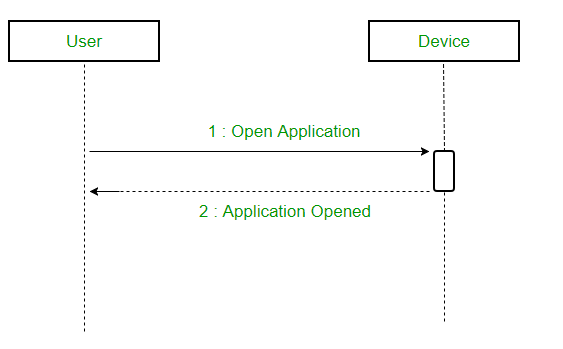
1. **Actors –** An actor in a UML diagram represents a type of role where it interacts with the system and its objects. It is important to note here that an actor is always outside the scope of the system we aim to model using the UML diagram.



1. **Lifelines –** A lifeline is a named element which depicts an individual participant in a sequence diagram.



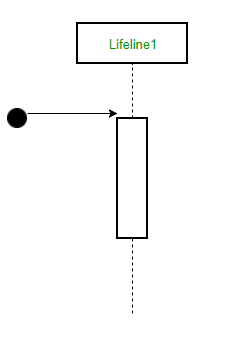
1. **Messages –** Communication between objects is depicted using messages. The messages appear in a sequential order on the lifeline. We represent messages using arrows. Lifelines and messages form the core of a sequence diagram. Messages can be broadly classified into the following **categories**: 
2. **Synchronous messages –** A synchronous message waits for a reply before the interaction can move forward. The sender waits until the receiver has completed the processing of the message. The caller continues only when it knows that the receiver has processed the previous message.



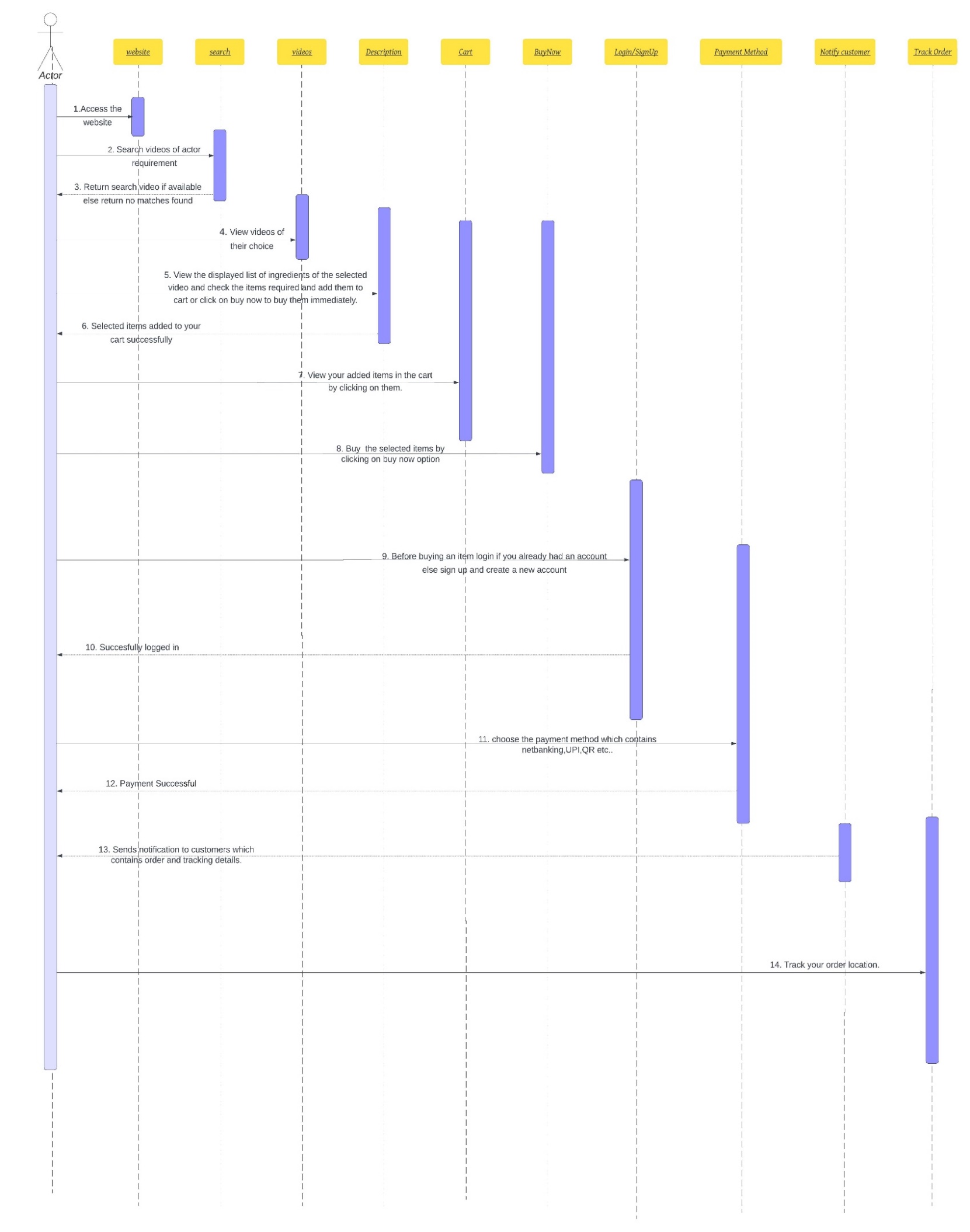
1. **Asynchronous Messages –** An asynchronous message does not wait for a reply from the receiver. The interaction moves forward irrespective of the receiver processing the previous message or not. We use a lined arrow head to represent an asynchronous message.



1. **Found Message –** A Found message is used to represent a scenario where an unknown source sends the message. It is represented using an arrow directed towards a lifeline from an end point. For example: Consider the scenario of a hardware failure.



Sequence Diagram



* 1. **Activity Diagram**

**Notations:**

1. **Initial State –** The starting state before an activity takes place is depicted using the initial state.

UML-State-Diagram

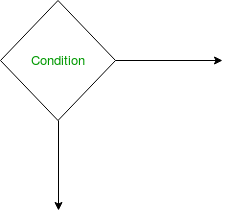
1. **Action or Activity State –** An activity represents execution of an action on objects or by objects. We represent an activity using a rectangle with rounded corners. Basically any action or event that takes place is represented using an activity.

UML-Activity-Diagram

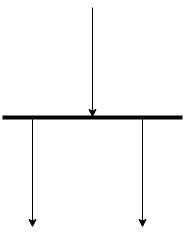
1. **Action Flow or Control flows –** Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another.

UML-Object-Diagram

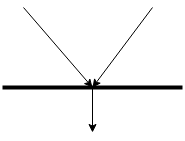
1. **Decision node and Branching –** When we need to make a decision before deciding the flow of control, we use the decision node.



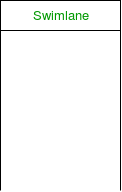
1. **Fork –** Fork nodes are used to support concurrent activities. Rectangular bar to represent a Fork notation with incoming arrow from the parent activity state and outgoing arrows towards the newly created activities.



1. **Join –** Join nodes are used to support concurrent activities converging into one. For join notations we have two or more incoming edges and one outgoing edge.



1. **Swimlanes –** We use swimlanes for grouping related activities in one column. Swimlanes group related activities into one column or one row. Swimlanes can be vertical and horizontal.



* 1. **State Chart Diagram**
  2. Initial state **–** We use a black filled circle represent the initial state of a System or a class.

UML-State-Diagram

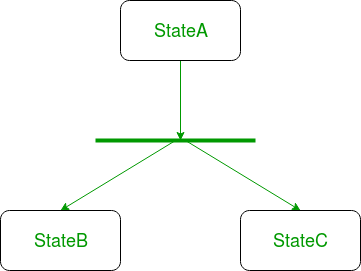
* 1. Transition **–** We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labelled with the event which causes the change in state.

UML-State-Diagram

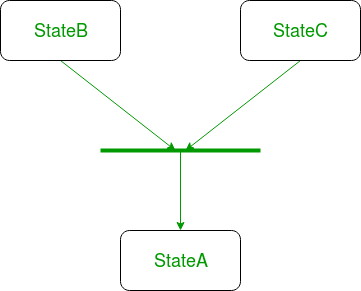
* 1. **State –** We use a rounded rectangle to represent a state. A state represents the conditions or circumstances of an object of a class at an instant of time.

UML-State-Diagram

1. Fork **–** We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent state and outgoing arrows towards the newly created states. We use the fork notation to represent a state splitting into two or more concurrent states.



1. **Join –** We use a rounded solid rectangular bar to represent a Join notation with incoming arrows from the joining states and outgoing arrow towards the common goal state. We use the join notation when two or more states concurrently converge into one on the occurrence of an event or events.



1. **Final state –** We use a filled circle within a circle notation to represent the final state in a state machine diagram.

UML-State-Diagram