Q1. What is SDLC?

Ans. SDLC is a structure imposed on the development of a software product that defines the processor planning, implementation, testing, documentation, deployment, and ongoing maintenance and support.

Q2. What is software testing?

Ans. Testing is the process of evaluating a system or its component(s) with the intent to find that whether it satisfies the specified requirements or not.

- This activity results in the actual, expected and difference between their results.
- In simple words testing is executing a system in order to identify any gaps, errors or missing requirements in contrary to the actual desire or requirements
- Software Testing is a process used to identify the correctness, completeness, and quality of developed computer software.

Q3. What is agile methodology?

Ans. Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product. Agile Methods break the product into small incremental builds.

- These builds are provided in iterations.
- Each iteration typically lasts from about one to three weeks.
- Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing.
- At the end of the iteration a working product is displayed to the customer and important stakeholders.

Q4. What is SRS?

Ans. • A software requirements specification (SRS) is a complete description of the behavior of the system to be developed.

- It includes a set of use cases that describe all of the interactions that the users will have with the software.

- Use cases are also known as functional requirements. In addition to use cases, the SRS also contains nonfunctional (or supplementary) requirements.

- Non-functional requirements are requirements which impose constraints on the design or implementation (such as performance requirements, quality standards, or design constraints).

- Recommended approaches for the specification of software requirements are described by IEEE 830-1998.

- This standard describes possible structures, desirable contents, and qualities of a software requirements specification.

Q5. What is oops?

Ans. Identifying objects and assigning responsibilities to these objects.

- Objects communicate to other objects by sending messages.

- Messages are received by the methods of an object

- An object is like a black box.

- The internal details are hidden.

- Object is derived from abstract data type

- Object-oriented programming has a web of interacting objects, each house-keeping its own state.

- Objects of a program interact by sending messages to each other.

Q6. Write Basic Concepts of oops?

Ans. Object

- Class

- Encapsulation

- Inheritance

- Polymorphism

    - Overriding

    - Overloading

- Abstraction

Q7. What is object?

Ans. An object represents an individual, identifiable item, unit, or entity, either real or abstract, with a well-defined role in the problem domain.

- An "object" is anything to which a concept applies.
- This is the basic unit of object oriented programming (OOP).
- That is both data and function that operate on data are bundled as a unit called as object.

Q8. What is class?

Ans. When you define a class, you define a blueprint for an object.

- This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.
- A class represents an abstraction of the object and abstracts the properties and behavior of that object.
- Class can be considered as the blueprint or definition or a template for an object and describes the properties and behavior of that object, but without any actual existence.
- An object is a particular instance of a class which has actual existence and there can be many objects (or instances) for a class.
- In the case of a car or laptop, there will be a blueprint or design created first and then the actual car or laptop will be built based on that. We do not actually buy these blueprints but the actual objects.

Q9. **What is encapsulation?**

Ans. Encapsulation is the practice of including in an object everything it needs hidden from other objects. The internal state is usually not accessible by other objects.

- Encapsulation in Java is the process of wrapping up of data (properties) and behavior (methods) of an object into a single unit; and the unit here is a Class (or interface).
- Encapsulate in plain English means to enclose or be enclosed in or as if in a capsule. In Java, a class is the capsule (or unit).
- In Java, everything is enclosed within a class or interface, unlike languages such as C and C++, where we can have global variables outside classes.
- Encapsulation enables data hiding, hiding irrelevant information from the users of a class and exposing only the relevant details required by the user.

- We can expose our operations hiding the details of what is needed to perform that operation.

- We can protect the internal state of an object by hiding its attributes from the outside world (by making it private), and then exposing them through setter and getter methods. Now modifications to the object internals are only controlled through these methods.

Q10. What is inheritance?

Ans. Inheritance means that one class inherits the characteristics of another class. This is also called a "is a" relationship.

- One of the most useful aspects of object-oriented programming is code reusability. As the name suggests Inheritance is the process of forming a new class from an existing class that is from the existing class called as base class, new class is formed called as derived class.

- This is a very important concept of object-oriented programming since this feature helps to reduce the code size.

- Inheritance describes the relationship between two classes. A class can get some of its characteristics from a parent class and then add unique features of its own.

- In general, Java supports single-parent, multiple-children inheritance and multilevel inheritance (Grandparent-> Parent -> Child) for classes and interfaces. Java supports multiple inheritances (multiple parents, single child) only through interfaces.

- In a class context, inheritance is referred to as implementation inheritance, and in an interface context, it is also referred to as interface inheritance.

There are five types of inheritance:

1.Single Inheritance

2.Multiple Inheritance

3.Multilevel inheritance

4.Hybrid Inheritance

5.Hierarchical Inheritance

Q11. What is polymorphism?

Ans. Polymorphism means "having many forms".

- It allows different objects to respond to the same message in different ways, the response specific to the type of the object.

• The most important aspect of an object is its behaviour (the things it can do). A behavior is initiated by sending a message to the object (usually by calling a method).

• The ability to use an operator or function in different ways in other words giving different meaning or functions to the operators or functions is called polymorphism.

• The ability to change form is known as polymorphism.

There are two types:

1.Overloading

2.Overriding

Q12. Write SDLC phases with basic introduction?

Ans. There are six phases:

1. Requirements Collection/Gathering-Establish Customer Needs

2. Analysis-Model and Specify the requirements

3. Design-Model and Specify a Solution

4. Implementation-Construct a Solution in Software

5. Testing-Validate the solution against the requirements

6. Maintenance-Repair defects and adapt the solution to the new requirements

1. Requirements Collection/Gathering:

- Features
- Usage scenarios
- Although requirements may be documented in written form, they may be incomplete unambiguous, or even incorrect.
- Requirements will Change!
- Inadequately captured or expressed in the first place
- User and business needs change during the project
- Validation is needed throughout the software lifecycle, not only when the “final system” is delivered.
- Build constant feedback into the project plan

- Plan for change
- Early prototyping [e.g., UI] can help clarify the requirements
- Functional and Non-Functional
- Requirements definitions usually consist of natural language, supplemented by (e.g., UML) diagrams and tables.

Types of Requirements:

- **Functional Requirements:** describe system services or functions.
    - Compute sales tax on a purchase
    - Update the database on the server
- **Non-functional Requirements:** are constraints on the system or the development process.

- Non-functional requirements may be more critical than functional requirements.
- If these are not met, the system is useless!

2. Analysis Phase:

- The analysis phase defines the requirements of the system, independent of how these requirements will be accomplished.
- This phase defines the problem that the customer is trying to solve.
- The deliverable result at the end of this phase is a requirement document.
- Ideally, this document states in a clear and precise fashion what is to be built.
- This analysis represents the “what” phase.
- The requirement documentaries to capture the requirements from the customer's perspective by defining goals.
- This phase starts with the requirement document delivered by the requirement phase and maps the requirements into architecture.
- The architecture defines the components, their interfaces and behaviors.
- The deliverable design document is the architecture.
- This phase represents the “how” phase.
- Details on computer programming languages and environments, machines, packages,

application architecture, distributed architecture layering, memory size, platform, algorithms, data structures, global type definitions, interfaces, and many other engineering details are established.

- The design may include the usage of existing components.

3. Design Phase:

- Design Architecture Document
- Implementation Plan
- Critical Priority Analysis
- Performance Analysis
- Test Plan
- The Design team can now expand upon the information established in the requirement document.
- The requirement document must guide this decision process.
- Analyzing the trade-offs of necessary complexity allows for many things to remain simple which, in turn, will eventually lead to a higher quality product. The architecture team also converts the typical scenarios into a test plan.

4. Implementation Phase:

• In the implementation phase, the team builds the components either from scratch or by composition.

• Given the architecture document from the design phase and the requirement document from the analysis phase, the team should build exactly what has been requested, though there is still room for innovation and flexibility.

• For example, a component may be narrowly designed for this particular system, or the component may be made more general to satisfy a reusability guideline.

- Implementation – Code

  • The implementation phase deals with issues of quality, performance, baselines, libraries, and debugging.

5. Testing Phase:

- Simply stated, quality is very important. Many companies have not learned that quality is important and deliver more claimed functionality but at a lower quality level.

- It is much easier to explain to a customer why there is a missing feature than to explain to a customer why the product lacks quality.
- A customer satisfied with the quality of a product will remain loyal and wait for new functionality in the next version.
- Quality is a distinguishing attribute of a system indicating the degree of excellence.
- Regression Testing
- Internal Testing
- Unit Testing
- Application Testing
- Stress Testing

6. Maintenance Phase:

- Software maintenance is one of the activities in software engineering, and is the process of enhancing and optimizing deployed software (software release), as well as fixing defects.
- Software maintenance is also one of the phases in the System Development Life Cycle (SDLC), as it applies to software development. The maintenance phase is the phase which comes after deployment of the software into the field.
- The developing organization or team will have some mechanism to document and track defects and deficiencies.
- configuration and version management
- reengineering (redesigning and refactoring)
- updating all analysis, design and user documentation
- Repeatable, automated tests enable evolution and refactoring

Maintenance is the process of changing a system after it has been deployed

- **Corrective Maintenance:** identifying and repairing defects
- **Adaptive Maintenance:** adapting the existing solution to the new platforms
- **Perfective Maintenance:** implementing the new requirements

Q13. Explain Phases of the waterfall model?

Ans. There are five phases of this model:

- Requirement/analysis
- Design
- Coding
- Testing
- Maintenance

Pros:

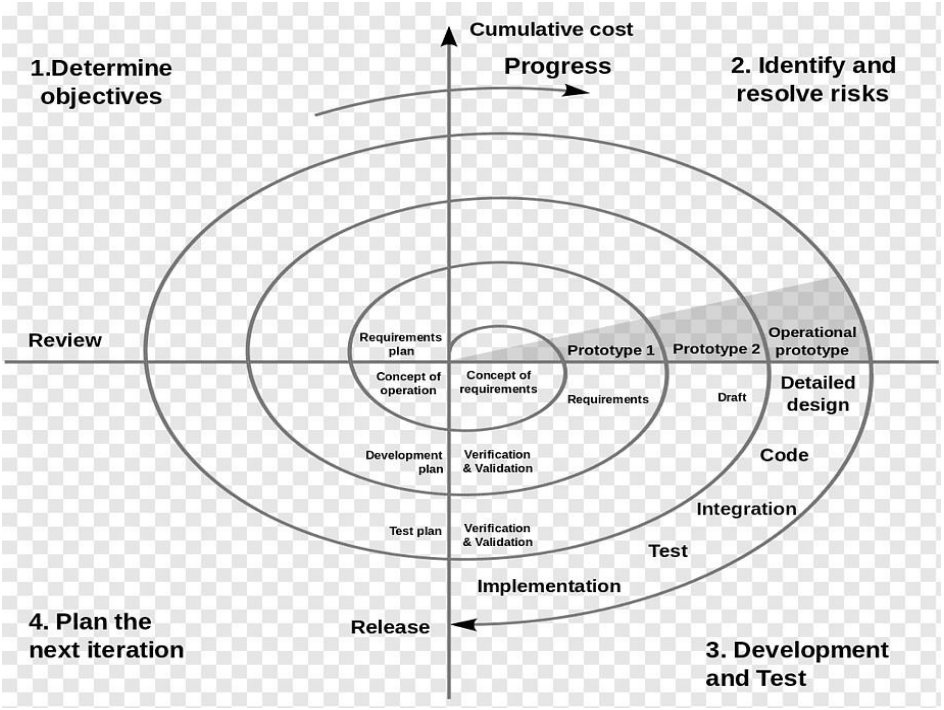
- Simple and easy to understand and use
- Easy to manage due to the rigidity of the model. Each phase has specific deliverables and a review process.
- Phases are processed and completed one at a time.
- Works well for smaller projects where requirements are very well understood.
- Clearly defined stages.
- Well understood milestones.

Cons:

- No working software is produced until late during the life cycle.
- High amounts of risk and uncertainty.
- Not a good model for complex and object-oriented projects.
- Not suitable for the projects where requirements are at a moderate to high risk of changing. So risk and uncertainty is high with this process model.
- Poor model for long and ongoing projects.
- Integration is done as a "big-bang. at the very end, which doesn't allow identifying any technological or business bottleneck or challenges early.

Q14. Write phases of spiral model?

Ans.



- The spiral model is a systems development lifecycle method used for risk management that combines the iterative development process model with elements of the waterfall model. The spiral model is used by software engineers and is favoured for large, expensive and complicated projects.

Pros:

- Changing requirements can be accommodated.
- Allows for extensive use of prototypes.
- Requirements can be captured more accurately.
- Users see the system early.
- Development can be divided into smaller parts and more risky parts can be developed earlier which helps better risk management.

Cons:

- Management is more complex.
- End of project may not be known early.

- Not suitable for small or low risk projects and could be expensive for small projects.
- Process is complex.
- Spiral may go indefinitely.
- Large number of intermediate stages requires excessive documentation.

Q15. Write agile manifesto principles?

Ans. **Individuals and interactions** - in agile development, self-organization and motivation are important, as are interactions like co-location and pair programming.

- **Working software** - Demo working software is considered the best means of communication with the customer to understand their requirement, instead of just depending on documentation.
- **Customer collaboration** - As the requirements cannot be gathered completely in the beginning of the project due to various factors, continuous customer interaction is very important to get proper product requirements.
- **Responding to change** - agile development is focused on quick responses to change and continuous development.

Pros:

- Is a very realistic approach to software development.
- Promotes teamwork and cross training.
- Functionality can be developed rapidly and demonstrated.
- Resource requirements are minimum.
- Suitable for fixed or changing requirements.
- Delivers early partial working solutions.
- Good model for environments that change steadily.
- Minimal rules, documentation easily employed.
- Enables concurrent development and delivery within an overall planned context.
- Little or no planning required.
- Easy to manage.

- Gives flexibility to developers.

Cons:

- Lack of documentation.
- Not suitable for handling complex dependencies.
- More risk of sustainability, maintainability and extensibility.
- An overall plan, an agile leader and agile PM practice is a must without which it will not work.
- Strict delivery management dictates the scope, functionality to be delivered, and adjustments to meet the deadlines.
- Depends heavily on customer interaction, so if customer is not clear, team can be driven in the wrong direction.
- There is very high individual dependency, since there is minimum documentation generated.
- Transfer of technology to new team members may be quite challenging due to lack of.

Q16. Explain working methodology of agile model and also write pros and cons?

Ans. Agile SDLC model is a combination of iterative and incremental process models with focus on process adaptability and customer satisfaction by rapid delivery of working software product.

- Agile Methods break the product into small incremental builds.
- These builds are provided in iterations.
- Each iteration typically lasts from about one to three weeks.
- Every iteration involves cross functional teams working simultaneously on various areas like planning, requirements analysis, design, coding, unit testing, and acceptance testing.
- At the end of the iteration a working product is displayed to the customer and important stakeholders.

Pros:

- In Agile methodology the delivery of software is unremitting.
- The customers are satisfied because after every Sprint working feature of the software is delivered to them.

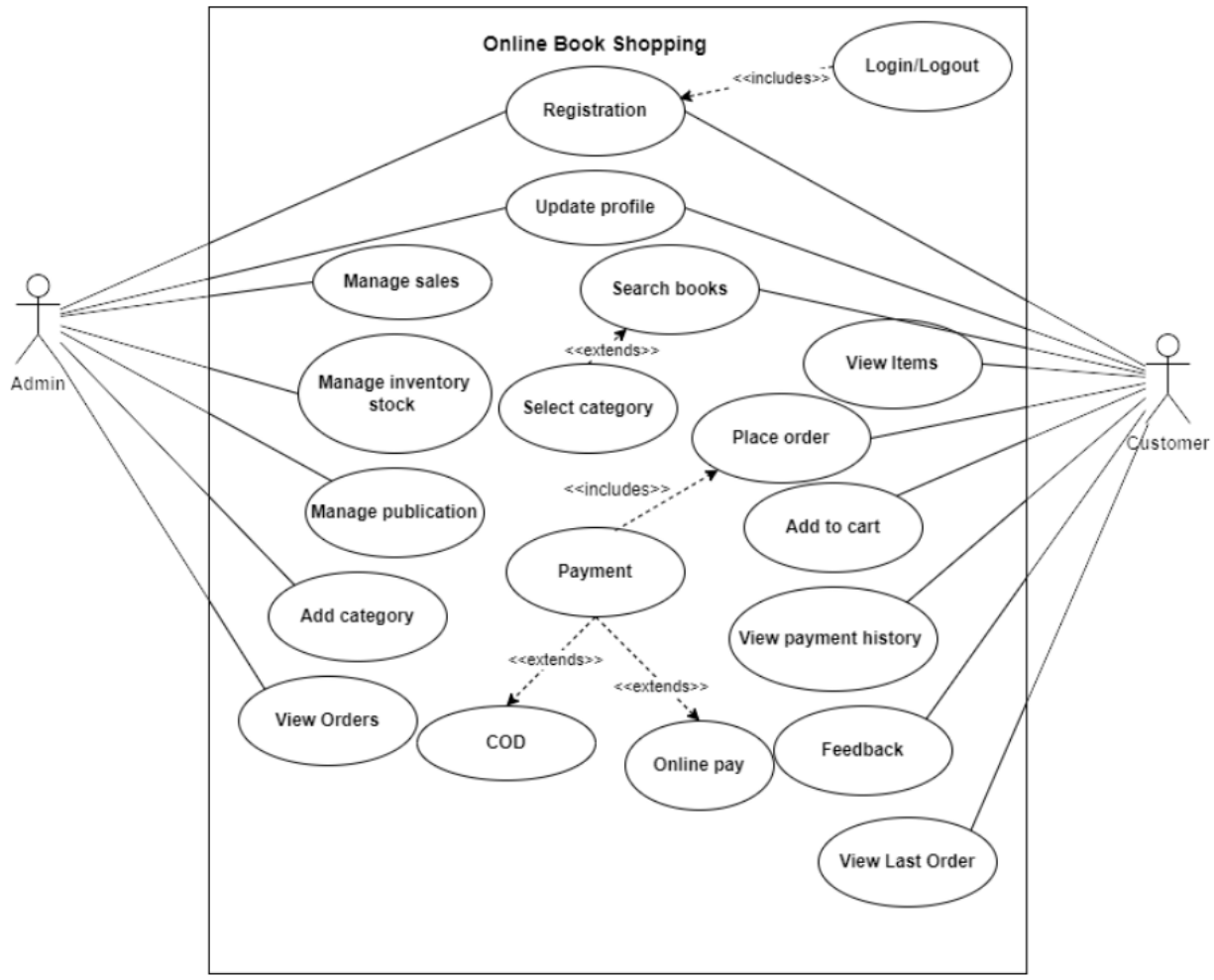
- Customers can have a look of the working feature which fulfilled their expectations.
- If the customers have any feedback or any change in the feature, then it can be accommodated in the current release of the product.
- In Agile methodology the daily interactions are required between the business people and the developers.
- In this methodology attention is paid to the good design of the product.
- Changes in the requirements are accepted even in the later stages of the development.
- An Agile/Scrum approach can improve organizational synergy by breaking down organizational barriers and developing a spirit of trust and partnership around organizational goals.

Cons:

- In Agile methodology the documentation is less.
- Sometimes in Agile methodology the requirement is not very clear hence it's difficult to predict the expected result.
- In few of the projects at the starting of the software development life cycle it's difficult to estimate the actual effort required.
- Because of the ever-evolving features, there is always a risk of the ever-lasting project.
- For complex projects, the resource requirement and effort are difficult to estimate.

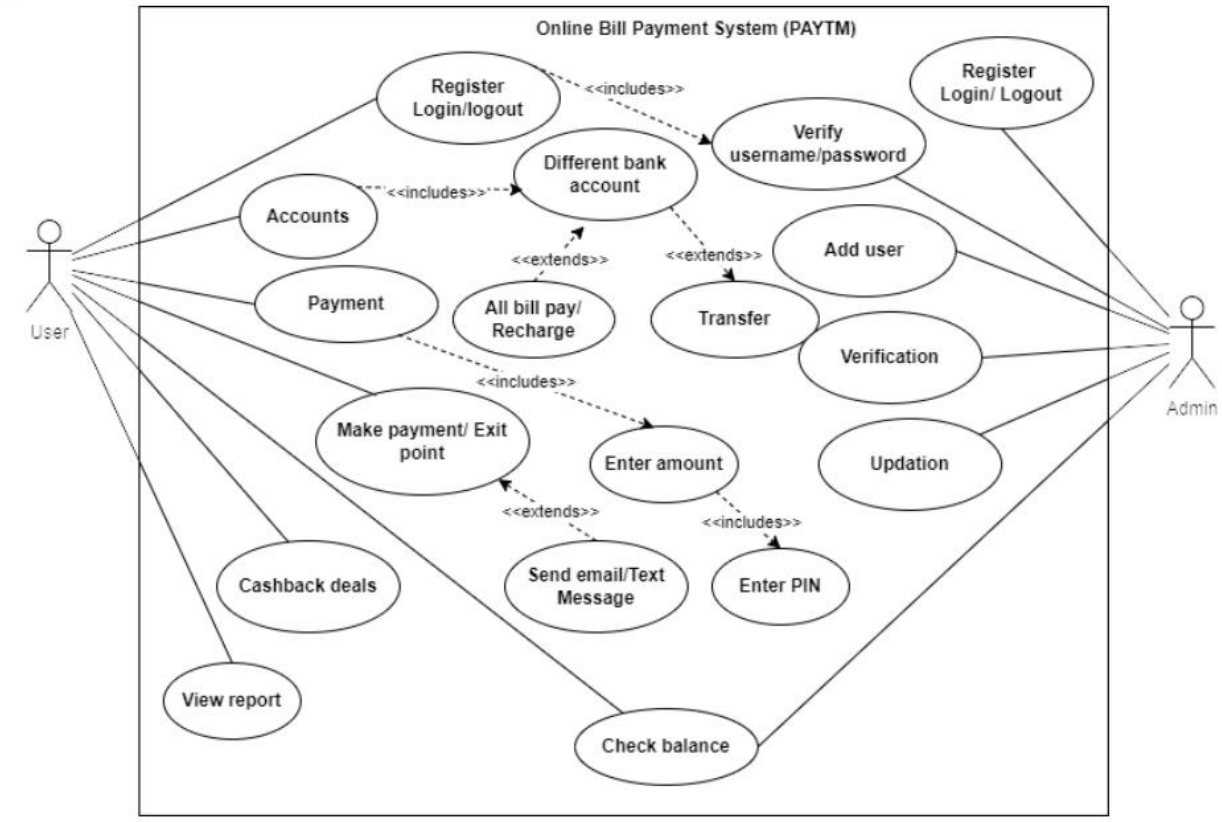
Q17. Draw Use case on Online book shopping?

Ans.



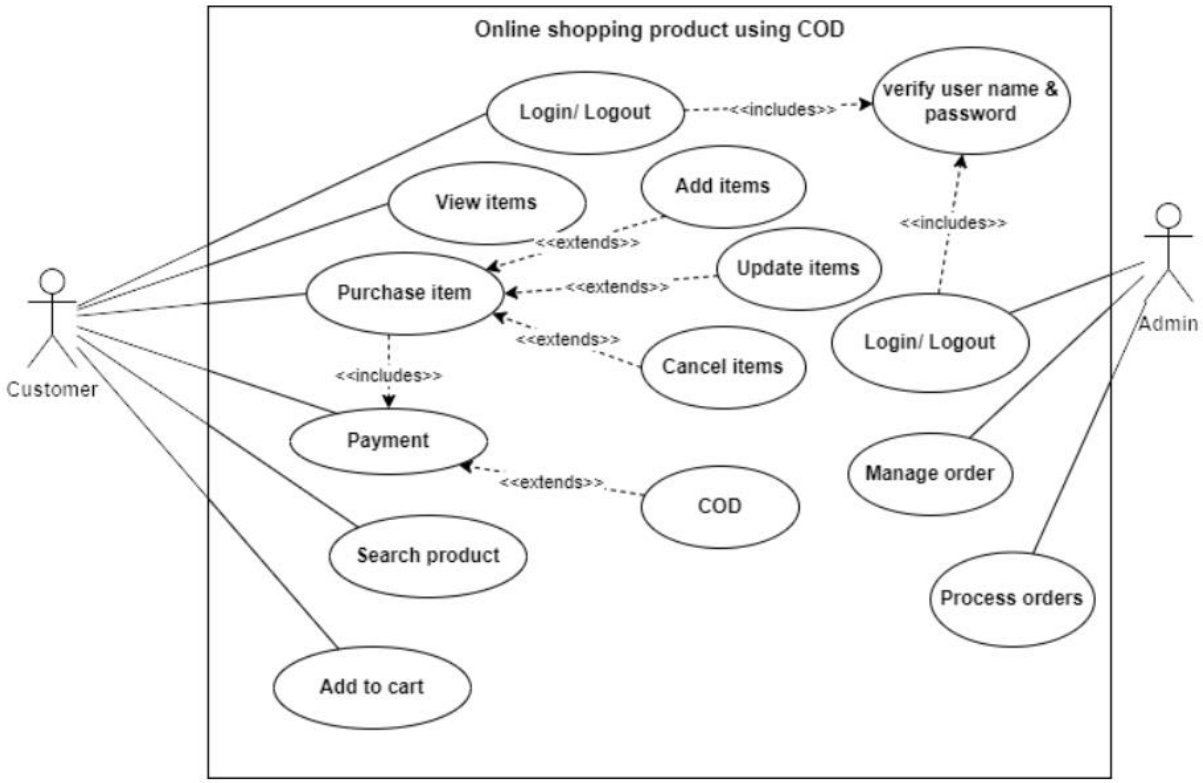
Q18. Draw Use case on online bill payment system (Paytm)?

Ans.



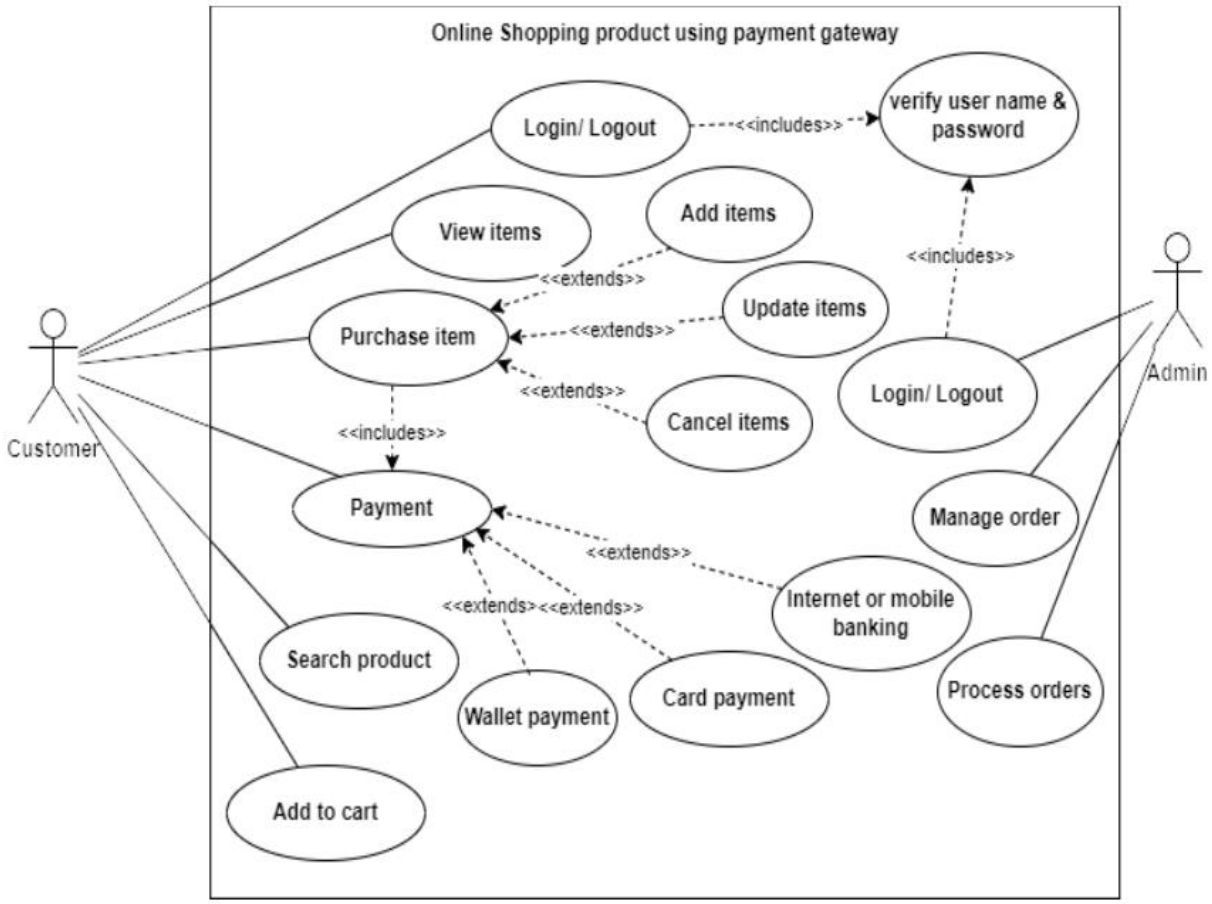
Q19. Draw use case on Online shopping product using COD?

Ans.



Q20. Draw use case on Online shopping product using payment gateway?

Ans.


Online Shopping product using payment gateway
Login/ Logout
<<includes>>
verify user name & password
View items
Add items
<<includes>>
<<extends>>
Update items
Purchase item
<<extends>>
Login/ Logout
<<extends>>
Cancel items
Customer
<<includes>>
Admin
Payment
Manage order
<<extends>>
Internet or mobile banking
<<extends><<extends>>
Search product
Wallet payment
Card payment
Process orders
Add to cart