Department of Computer Engineering

Indira College of Engineering and Management, Pune

ACADEMIC YEAR: 2024-25

A Project Report On

"UML Diagrams For Safe Pass Drive"

SUBMITTED TO SAVITRIBAI PHULE UNIVERSITY, PUNE

IN PARTIAL FULFILLMENT OF FOURTH YEAR BACHELOR OF ENGINEERING IN COMPUTER ENGINEERING

By:

Name	**Roll No**
Vaishnavi Mhamane	24133
Ajay Sonawane	24134
Sameer Kulkarni	24148
Siddhi More	24151

Under Guidance of:

Prof. Shraddha Suryawanshi

Department of Computer Engineering

Indira College of Engineering and Management, Pune

ACADEMIC YEAR: 2024-25

CERTIFICATE

THIS IS TO CERTIFY THAT

Name	**Roll No.**
Vaishnavi Mhamane	24133
Ajay Sonawane	24134
Sameer Kulkarni	24148
Siddhi More	24151

Group No 36 Division A Branch Computer Engineering has successfully completed the all UML Diagram for safe pass drive project titled as **"UML Diagrams For Safe Pass Drive".** This is a bonafide work carried out by them under the supervision of Prof. Shraddha Suryawanshi, in the partial fullfillment of the final year Bachelor of engineering (Choice Based Credit System) (2019 Course) Of Savitribai Phule Pune University.

Date:

Place: Pune

(Prof. Shraddha Suryawanshi)	(Dr. Soumitra Das)	(Dr.Nilesh Uke)
Project Guide	Head of Department	Principle

ACKNOWLEDGEMENT

It is great pleasure for me to acknowledge the assistance and contribution of number of individuals who helped me in drawing **UML Diagrams For Safe Pass Drive** First and foremost, I wish to record my gratitude and thanks to Prof. Shraddha Suryawanshi for his enthusiastic guidance and help in successful completion of Project work. I express my thanks to Prof. Soumitra Das (Head of Department), Prof. Sunil Rathod for their valuable guidance. I am also thankful to other teachers and non-teaching staff of Computer Engineering Department for their co-operation and help.

THANKING TO ALL OF YOU...

Index

Problem Statement:

Draw Use case, Class, State, Activity, Component Diagrams for Safe Pass Drive

Objectives:

The objective of drawing all UML diagrams is to provide a comprehensive, structured, and visual representation of the **system architecture**, **behaviour**, and **interactions**. Specifically, the objectives are:

- **Visualize System Structure**:
  - To clearly model the static structure of the system, including its components, classes, and relationships, ensuring a high-level understanding of how different parts of the system are organized.

- **Model System** behaviour:
  - To represent dynamic behaviour such as workflows, interactions, and state transitions, allowing stakeholders to understand how the system functions during runtime.

- **Ensure Completeness of Design**:
  - To cover all key aspects of the system by using different UML diagrams to represent various views (use cases, structural components, interactions, states, and deployment), ensuring that no critical detail is overlooked.

- **Facilitate Communication**:
  - To serve as a common language for communication among developers, stakeholders, and project managers, making it easier to explain and discuss the system's design and behaviour.

- **Enhance System Understanding and Maintenance**:
  - To provide a clear, organized model that can be referenced during the system's development and maintenance phases, making it easier to understand and modify the system as needed.

- **Identify Potential Design Issues Early**:
  - To detect potential design flaws, such as incorrect relationships or undefined behaviour, through careful analysis of the diagrams before actual coding begins.

- **Support Documentation**:
  - To create formal documentation of the system architecture and design that can be referenced by developers, testers, and future teams, ensuring long-term maintainability of the system.

1. Use Case Diagram

1. Introduction

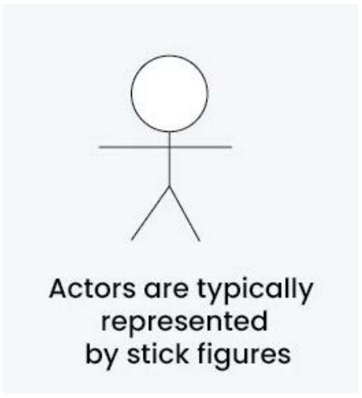
A Use Case Diagram is a type of Unified behaviour Language (UML) diagram that represents the interaction between actors (users or external systems) and a system under consideration to accomplish specific goals. It provides a high-level view of the system's functionality by illustrating the various ways users can interact with it.

Use Case Diagram Notations

UML notations provide a visual language that enables software developers, designers, and other stakeholders to communicate and document system designs, architectures, and behaviour in a consistent and understandable manner.

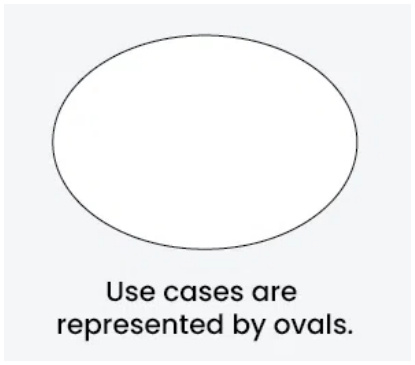
1. Actors

Actors are external entities that interact with the system. These can include users, other systems, or hardware devices. In the context of a Use Case Diagram, actors initiate use cases and receive the outcomes. Proper identification and understanding of actors are crucial for accurately modelling system behaviour.



2. Use Cases

Use cases are like scenes in the play. They represent specific things your system can do. In the online shopping system, examples of use cases could be “Place Order,” “Track Delivery,” or “Update Product Information”. Use cases are represented by ovals.

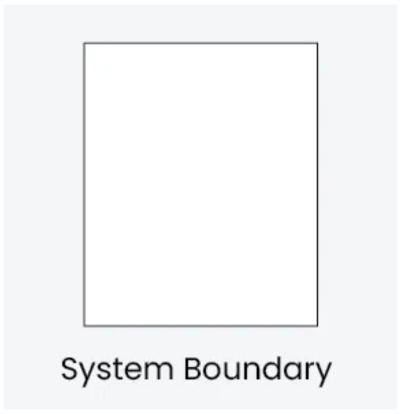


3. System Boundary

The system boundary is a visual representation of the scope or limits of the system you are modelling. It defines what is inside the system and what is outside. The boundary helps to establish a clear distinction between the elements that are part of the system and those that are external to it. The system boundary is typically represented by a rectangular box that surrounds all the use cases of the system.

Purpose of System Boundary:

- Scope Definition: It clearly outlines the boundaries of the system, indicating which components are internal to the system and which are external actors or entities interacting with the system.
- Focus on Relevance: By delineating the system's scope, the diagram can focus on illustrating the essential functionalities provided by the system without unnecessary details about external entities.



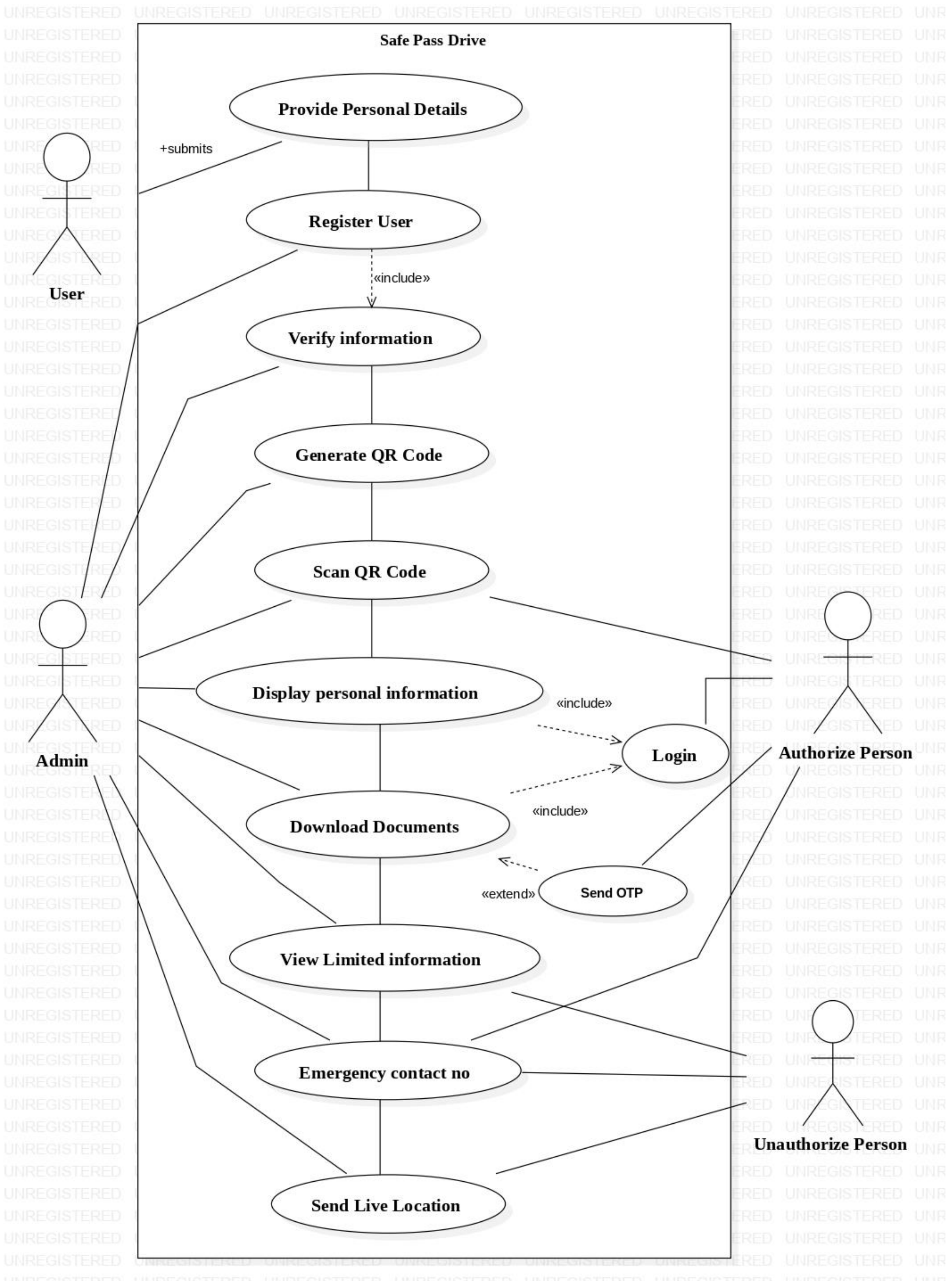


Figure: Use Case Diagram

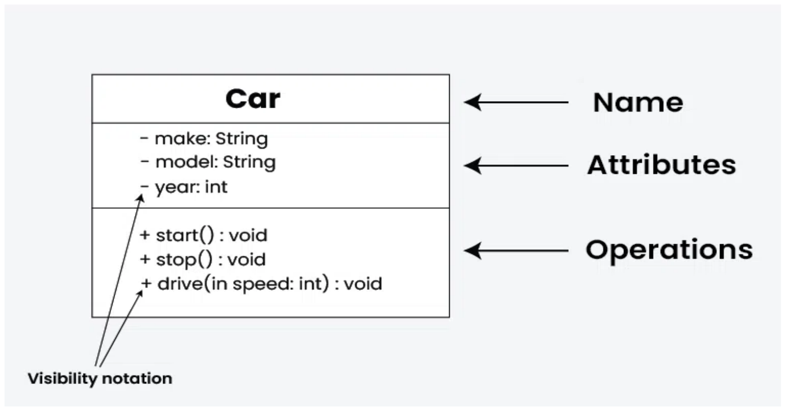
2. Class Diagram

1. Introduction

Class diagrams are a type of UML (Unified Modelling Language) diagram used in software engineering to visually represent the structure and relationships of classes within a system i.e. used to construct and visualize object-oriented systems. Class diagrams provide a high-level overview of a system's design, helping to communicate and document the structure of the software. They are a fundamental tool in object-oriented design and play a crucial role in the software development lifecycle.

UML Class Notation

Class notation is a graphical representation used to depict classes and their relationships in object-oriented modelling.



1. **Class Name:**
   The name of the class is typically written in the top compartment of the class box and is centered and bold.
2. **Attributes:**
   Attributes, also known as properties or fields, represent the data members of the class. They are listed in the second compartment of the class box and often include the visibility (e.g., public, private) and the data type of each attribute.
3. **Methods:**
   Methods, also known as functions or operations, represent the behaviour or functionality of the class. They are listed in the third compartment of the class box and include the visibility (e.g., public, private), return type, and parameters of each method.
4. **Visibility Notation:**
   Visibility notations indicate the access level of attributes and methods. Common visibility notations include:
   - + for public (visible to all classes)
   - - for private (visible only within the class)
   - # for protected (visible to subclasses)
   - ~ for package or default visibility (visible to classes in the same package

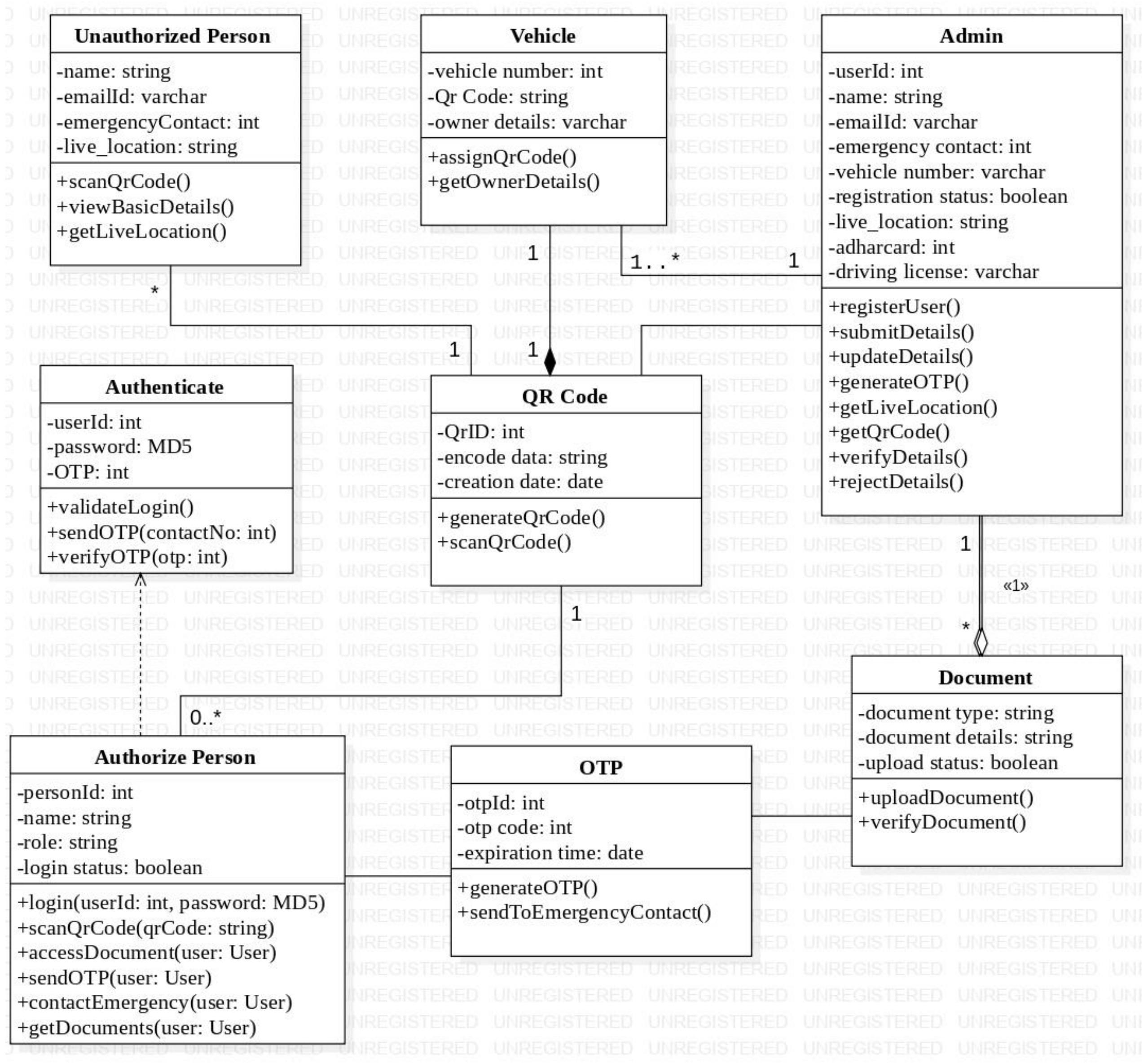


Figure: Class Diagram

3. Activity Diagram

1.Introduction

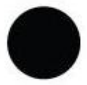
Activity Diagrams are used to illustrate the flow of control in a system and refer to the steps involved in the execution of a use case. We can depict both sequential processing and concurrent processing of activities using an activity diagram i.e an activity diagram focuses on the condition of flow and the sequence in which it happens.

- We describe what causes a particular event using an activity diagram.
- An activity diagram portrays the control flow from a start point to a finish point showing the various decision paths that exist while the activity is being executed.
- They are used in business and process modelling where their primary use is to depict the dynamic aspects of a system

Activity Diagram Notations

1. Initial State

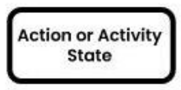
The starting state before an activity takes place is depicted using the initial state
A process can have only one initial state unless we are depicting nested activities. We use a black filled circle to depict the initial state of a system. For objects, this is the state when they are instantiated. The Initial State from the UML Activity Diagram marks the entry point and the initial Activity State.



Initial State

2. Action or Activity State

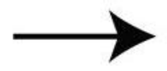
An activity represents execution of an action on objects or by objects. We represent an activity using a rectangle with rounded corners. Basically any action or event that takes place is represented using an activity.



Activity State

3. **Action Flow or Control flows**

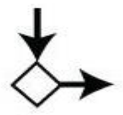
   Action flows or Control flows are also referred to as paths and edges. They are used to show the transition from one activity state to another activity state. An activity state can have multiple incoming and outgoing action flows. We use a line with an arrow head to depict a Control Flow. If there is a constraint to be adhered to while making the transition it is mentioned on the arrow.



Control Flow

4. **Decision node and Branching**

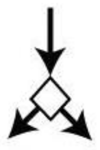
   When we need to make a decision before deciding the flow of control, we use the decision node. The outgoing arrows from the decision node can be labelled with conditions or guard expressions. It always includes two or more output arrows.



Decision node

5. **Guard**

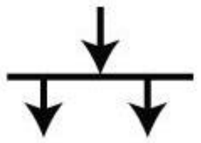
   A Guard refers to a statement written next to a decision node on an arrow sometimes within square brackets. The statement must be true for the control to shift along a particular direction. Guards help us know the constraints and conditions which determine the flow of a process.



Guard

6. **Fork**

   Fork nodes are used to support concurrent activities. When we use a fork node when both the activities get executed concurrently i.e. no decision is made before splitting the activity into two parts. Both parts need to be executed in case of a fork statement. We use a rounded solid rectangular bar to represent a Fork notation with incoming arrow from the parent activity state and outgoing arrows towards the newly created activities.



Fork

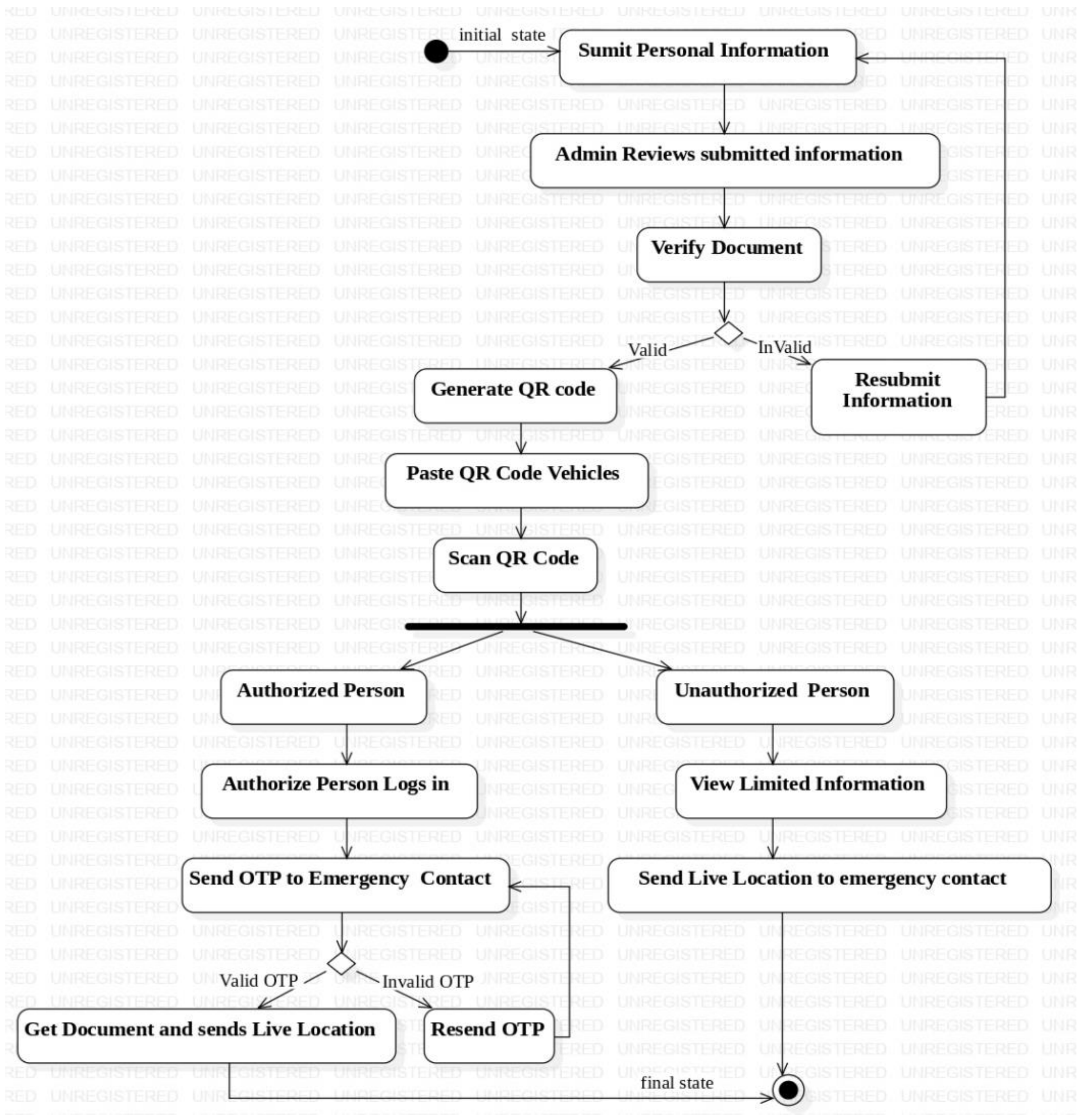


Figure: Activity Diagram

4. Component Diagram

1. Introduction

A Component-Based Diagram, often called a Component Diagram, is a type of structural diagram in the Unified Modelling Language (UML) that visualizes the organization and interrelationships of the components within a system. Component-Based Diagrams are widely used in system design to promote modularity, enhance understanding of system architecture**.**

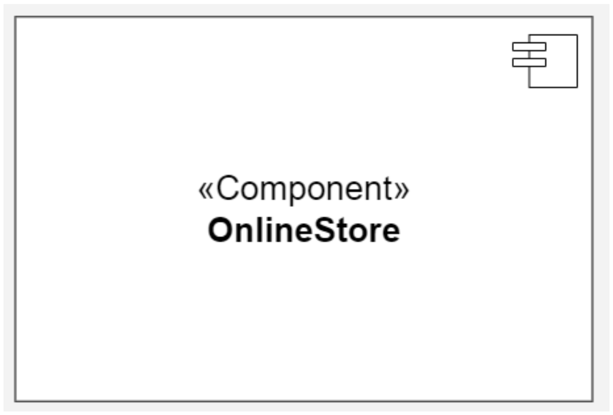
- Components are modular parts of a system that encapsulate implementation and expose a set of interfaces.

- These diagrams illustrate how components are wired together to form larger systems, detailing their dependencies and interactions.

Components of Component-Based Diagram

1. Component:

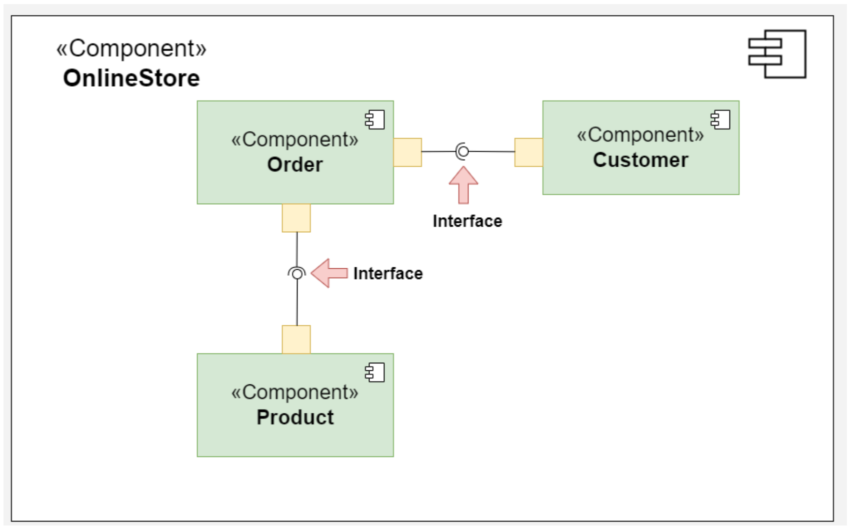
- **Role**: Represent modular parts of the system that encapsulate functionalities. Components can be software classes, collections of classes, or subsystems.
- **Symbol**: Rectangles with the component stereotype («component»).
- **Function**: Define and encapsulate functionality, ensuring modularity and reusability.



2. Interfaces:

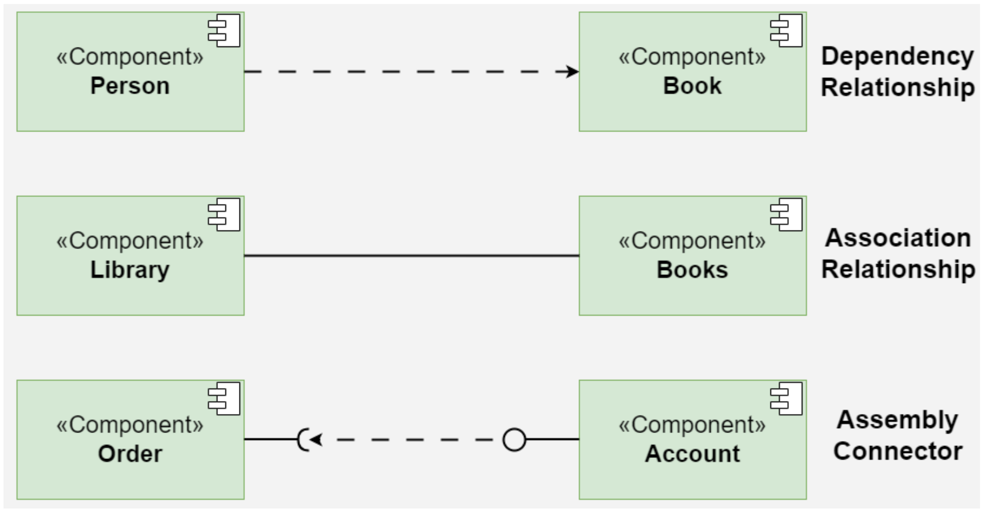
- **Role**: Specify a set of operations that a component offers or requires, serving as a contract between the component and its environment.
- **Symbol**: Circles (lollipops) for provided interfaces and half-circles (sockets) for required interfaces.

- **Function**: Define how components communicate with each other, ensuring that components can be developed and maintained independently.



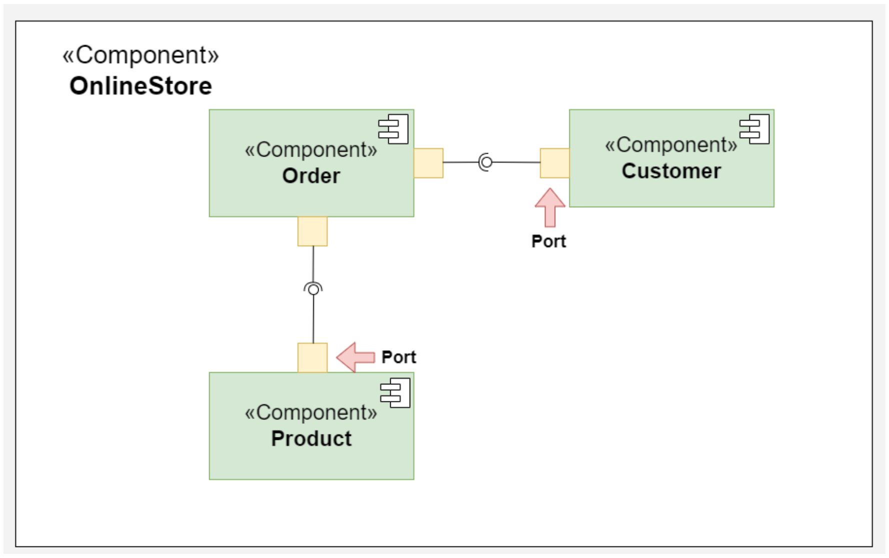
3. Relationships:

- **Role**: Depict the connections and dependencies between components and interfaces.
- **Symbol**: Lines and arrows.
  - Dependency (dashed arrow): Indicates that one component relies on another.
  - Association (solid line): Shows a more permanent relationship between components.
  - Assembly connector: Connects a required interface of one component to a provided interface of another.
- **Function**: Visualize how components interact and depend on each other, highlighting communication paths and potential points of failure.



4. Ports:

- **Role:** Represent specific interaction points on the boundary of a component where interfaces are provided or required.
- **Symbol:** Small squares on the component boundary.
- **Function:** Allow for more precise specification of interaction points, facilitating detailed design and implementation.



5. Artifacts:

- **Role:** Represent physical files or data that are deployed on nodes.
- **Symbol:** Rectangles with the artifact stereotype («artifact»).
- **Function:** Show how software artifacts, like executables or data files, relate to the components.

6. Nodes:

- **Role:** Represent physical or virtual execution environments where components are deployed.
- **Symbol:** 3D boxes.
- **Function:** Provide context for deployment, showing where components reside and execute within the system's infrastructure.

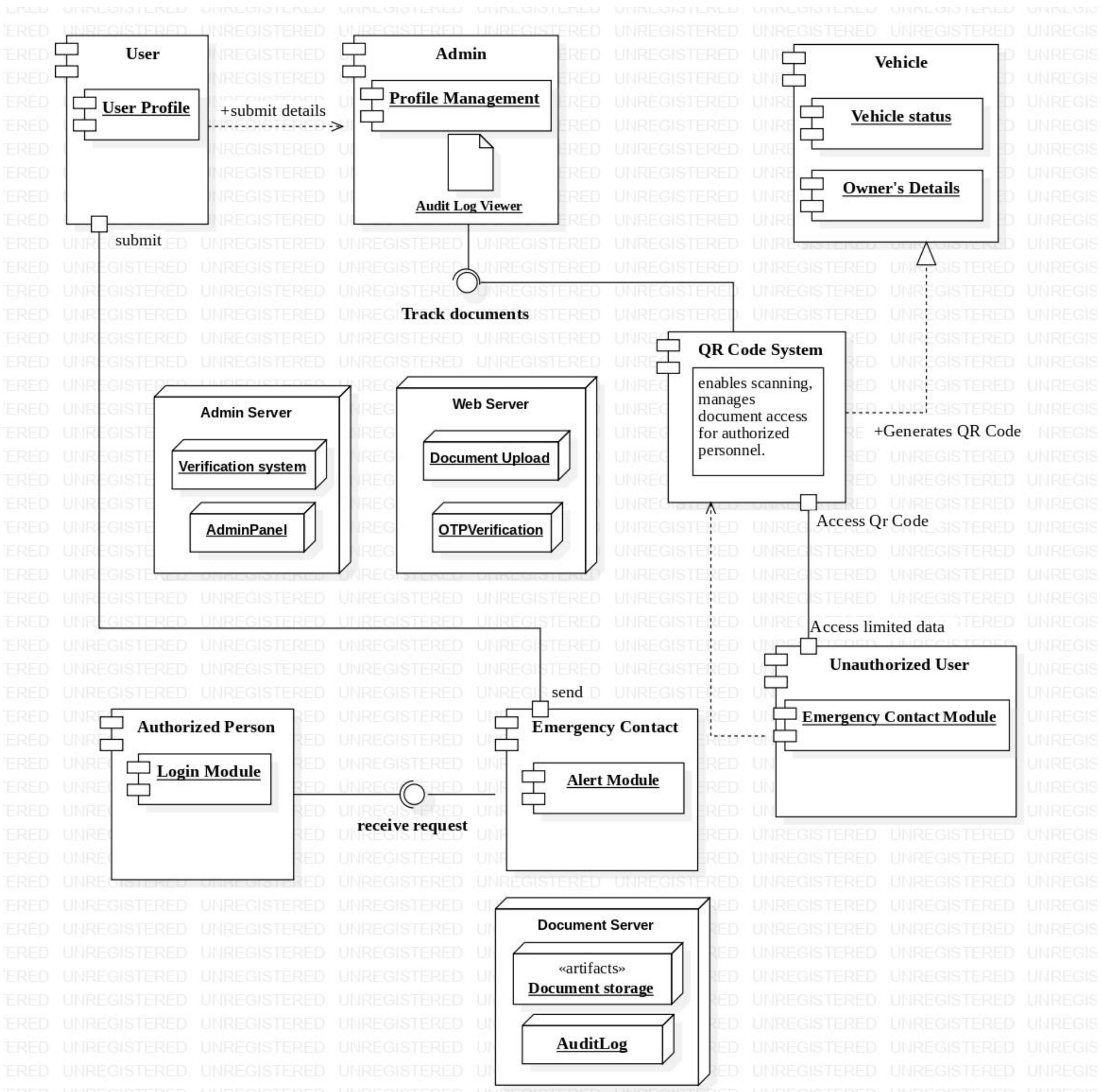


Figure: Component Diagram

5. State Diagram

1. Introduction

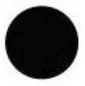
A state diagram is used to represent the condition of the system or part of the system at finite instances of time. It is a behaviour diagram and it represents the behaviour using finite state transitions.

- State Machine diagrams are also referred to as State Machines Diagrams and State-Chart Diagrams.
- These terms are often used interchangeably. So simply, a state machine diagram is used to model the dynamic behaviour of a class in response to time and changing external stimuli.
- We can say that each and every class has a state but we don't model every class using State Machine diagrams.
- We prefer to model the states with three or more states.

Basic components and notations of a State Machine diagram

1. Initial state

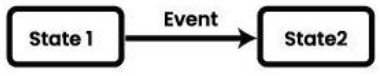
We use a black filled circle represent the initial state of a System or a Class.



Initial State

2. Transition

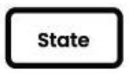
We use a solid arrow to represent the transition or change of control from one state to another. The arrow is labelled with the event which causes the change in state.



Transition

3. State

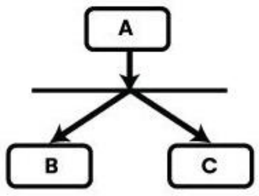
We use a rounded rectangle to represent a state. A state represents the conditions or circumstances of an object of a class at an instant of time.



State

4. Join

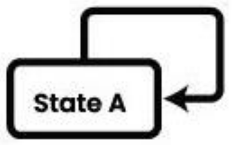
We use a rounded solid rectangular bar to represent a Join notation with incoming arrows from the joining states and outgoing arrow towards the common goal state. We use the join notation when two or more states concurrently converge into one on the occurrence of an event or events.



Fork

5. Self Transition

We use a solid arrow pointing back to the state itself to represent a self transition. There might be scenarios when the state of the object does not change upon the occurrence of an event. We use self transitions to represent such cases.



Self Transition

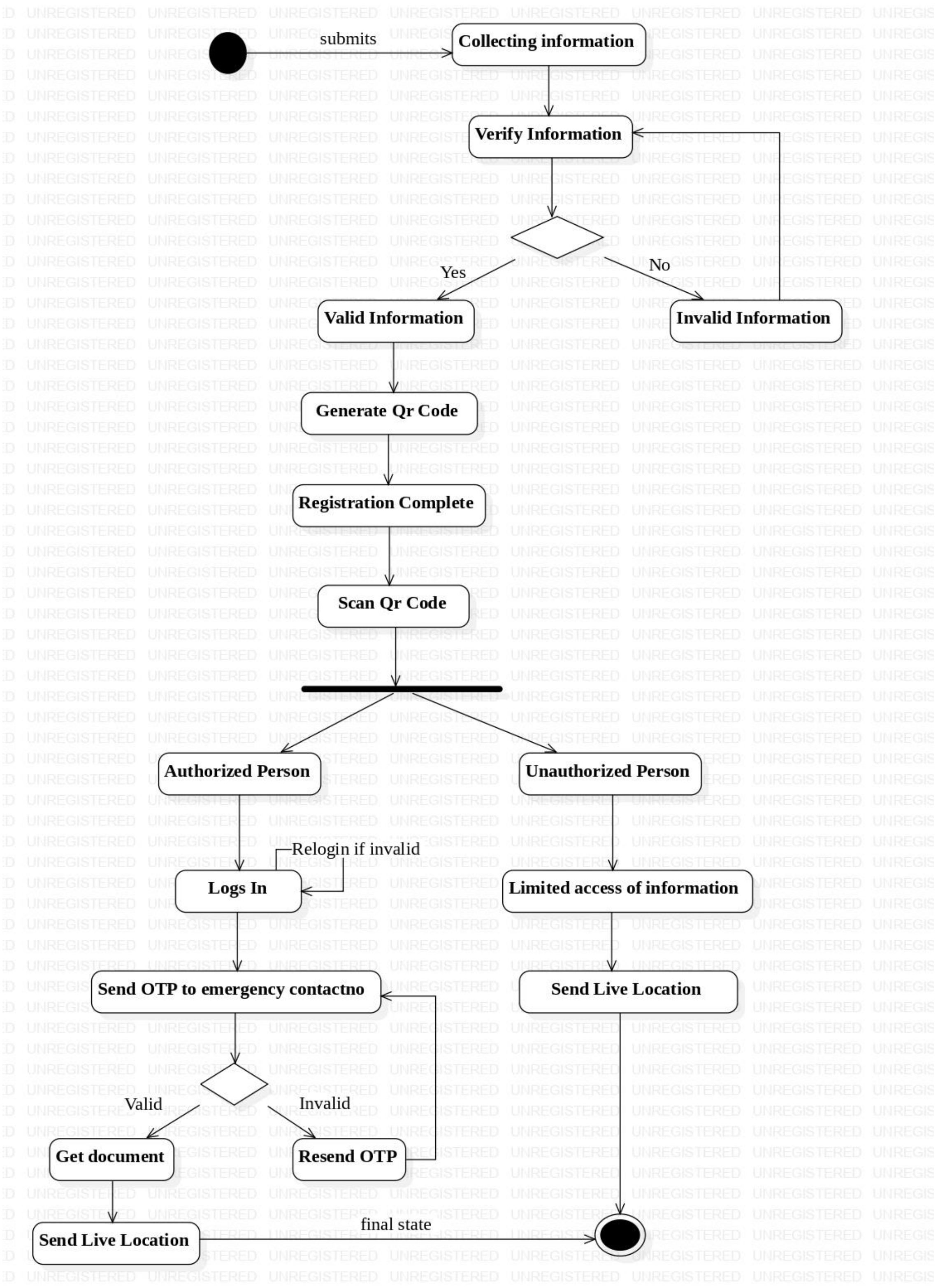


Figure: State Diagram

Software Requirements

SR.NO	Software Component	Details
1	Operating System	Windows 10 and above
2	Browser	Chrome
3	Editor	MS Word

Hardware Requirements

SR.NO	Component	Details
1	Processor	Core i5 and above
2	Memory	RAM 8GB, HDD:512 GB

Conclusion:

Drawing all UML diagrams for a software system provides a clear, structured, and holistic view of the system's architecture and behaviour. By modelling the system using different UML diagrams, we can:

- Ensure a comprehensive understanding of both the static and dynamic aspects of the system.
- Facilitate effective communication among stakeholders, including developers, project managers, and clients, by providing a common visual language.
- Identify potential design flaws and inconsistencies early in the development process, reducing risks and improving system quality.
- Support system documentation and future maintenance by providing a detailed blueprint that can be referenced throughout the software lifecycle.

References:

- https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/
- https://www.javatpoint.com/uml-class-diagram