

**SVKM’s NMIMS**

**School of Technology Management and Engineering, Kharghar, Navi-Mumbai-410210**

**Department of Electronics & Telecommunication**

**Academic Year-2020-21**

**Programme: B.Tech(Integrated)- Computer**

**Year-II, Semester- IV**

**Course: Software Design with UML**

**COURSE FACULTY**

**Tejaswini Chavan**

**Index**

|  |  |  |
| --- | --- | --- |
| **Sr.No** | **Title** | **Page No.** |
| **1** | **Student Outcomes/ Programme Outcomes** | **3** |
| **2** | **Course Outcomes** | **4** |
| **3** | **Mapping of Course Outcomes with Student Outcomes/ Programme Outcomes** | **4** |
| **4** | **Mapping of Lab Experiments to Course Outcomes** | **5** |
| **5** | **Lab Manual** | **6** |

**Student Outcomes**

Graduates of the Computer Engineering program will have an ability to:

1. Identify, formulate, and solve complex engineering problems by applying principles of engineering, science, and mathematics.

2. Apply engineering design to produce solutions that meet specified needs with consideration of public health, safety, and welfare, as well as global, cultural, social, environmental, and economic factors.

3. Communicate effectively with a range of audiences.

4. Recognize ethical and professional responsibilities in engineering situations and make informed judgments, which must consider the impact of engineering solutions in global, economic, environmental, and societal contexts.

5. Function effectively on a team whose members together provide leadership, create a collaborative and inclusive environment, establish goals, plan tasks, and meet objectives.

6. Develop and conduct appropriate experimentation, analyze and interpret data, and use engineering judgment to draw conclusions.

7. Acquire and apply new knowledge as needed, using appropriate learning strategies.

**Course Outcomes:**

|  |  |
| --- | --- |
| CO-1 | Understand unified process. |
| CO-2 | Design UML for the given problem statement. |
| CO-3 | Analyse the product using testing methodologies |
| CO-4 | Apply project management skills to manage the project. |

**Mapping of Course Outcomes with student Outcomes:**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Mapping | SO-1 | SO-2 | SO-3 | SO-4 | SO-5 | SO-6 | SO-7 |
| CO-1 | M |  |  |  |  |  |  |
| CO-2 |  | H |  |  | M |  |  |
| CO-3 |  |  |  |  |  | H |  |
| CO-4 |  |  |  | M | H |  | M |

**S****VKM’s NMIMS**

**School of Technology Management & Engineering**

**Computer Engineering Department**

Program: B.Tech CSBS

**Course: Object Oriented Software Engineering**

**List of Experiments**

**Faculty:** Prof. Tejaswini Chavan

|  |  |
| --- | --- |
|  |  |
| CO-1 | Understand unified process. |
| CO-2 | Design UML for the given problem statement. |
| CO-3 | Analyse the product using testing methodologies. |
| CO-4 | Apply project management skills to manage the project. |

|  |  |  |  |
| --- | --- | --- | --- |
| **Exp No.** | **Title** | **Prerequisite\*** | **CO#** |
| [1](#exp1) | Learning the StarUML environment and Identifying a problem statement for preparation of SRS document using structured approach and understand different phases of unified process of object oriented software engineering. | Concepts of Software Engineering Model | CO1 |
| [2.](#exp2) | Modeling UML Use Case diagrams and capturing Use Case scenarios. | Concepts of Actor, Use Case and Relationships | CO2 |
| [3.](#exp3) | To draw the behavioral view diagram: Sequence diagram, Collaboration diagram | Determine the desired flow of action and their interaction with each other | CO2 |
| [4](#exp4). | To draw the behavioral view diagram: Activity diagram | Concepts of Actor, Use Case and Relationships | CO2 |
| [5.](#exp5) | To draw the structural view diagram: Class diagram, object diagram. Learn the object-oriented analysis phase by understanding the methods of class elicitation and finding the classes in an object-oriented system. | Concepts of objects and class | CO3 |
| [6.](#exp6) | To draw the state chart diagram. | Concepts of behavioral model, states | CO3 |
| [7.](#exp7) | To draw the Component diagram | Concepts of Class Diagram | CO3 |
| [8.](#exp8) | To draw the Deployment diagram | Requirement gathering, infrastructure of the client site | CO4 |
| [9.](#exp9) | Coding for any Two module of your project selected according to the designs which already designed and analyzed in EXP.1-7 | Code, understanding about unit testing, integration testing | CO4 |
| [10.](#exp10) | To understand and develop test cases for object-oriented systems. | Function point, COCOMO model | CO4 |

**PART A**

**EXPERIMENT NO. 1**

**A.1 Aim: - Learning the Star UML environment.**

**A.2 Prerequisite**

**1. Concepts of Software Engineering, Process Model**

**A.3 Outcome**

After successful completion of this training students will be able to use Star UML for creating the UML model with respect to given case study.

Design solution using unified modeling language.

**A.4 Task:**

**Every student needs to follow following steps and record the findings in appropriate section of PART B**

1. Download **Star UML** software in your machine and prepare necessary documentation for doing hands on with Star UML.
2. Select the appropriate case study to design **UML diagrams**.
3. Identify the **Scope** of the problems selected
4. Identify the **end user** of the solution
5. Identify the **functional** requirements of the project
6. Identify the **non-functional** requirements of the project
7. Identify the **feasibility** of the project (Economical feasibility)
8. Frame the **final** problem statement.
9. List down the list of **user requirements, system requirements**.
10. Identify the **ambiguities, inconsistencies, incompleteness** from the requirements gathered.
11. List down the development plan for the selected problem statements- **SDLC (Software development life cycle)**

**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

*(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded on Blackboard LMS or emailed to the concerned Lab in charge Faculties at the end of practical; in case Blackboard is not accessible)*

**B.1 About Star UML**

**StarUML**

StarUML is a software engineering tool for system modelling using Unified Modelling Language, System Modelling Language, and classical modelling notations. It is a visual language representing the working of a software system that can help non-programmers understand without getting lost.

**StarUML environment**

StarUML software can be sued by developers, analysts, testers, project managers, technical writers and end-users. StarUML supports class diagrams, use-case diagrams, activity diagrams, sequence diagrams, state diagrams, data-flow diagrams, flowchart diagrams, etc.

**How did StarUML come into existence?**

**Software Development Lifecycle (SDLC)**

SDLC is a process that defines framework activities to develop a good quality software product and it is used by most software industries. There are five stages in SDLC – Communication, planning, modelling, construction and deployment.

Following are the different phases it goes through -

1. Requirements Analysis
2. Feasibility Study
3. Design Plan
4. Software Development
5. In-Depth Software Testing
6. Software Deployment
7. Product Maintenance and Enhancement

**Software Requirements Specification Document (SRS Document)**

An SRS document describes what the software will do and how it will be expected to perform. It’s considered one of the initial stages of SDLC. It describes the functionality the product needs to fulfil all stakeholders (business, users) needs.

Now, stakeholders or end-users won’t understand if the developers show the raw code to them. The software is developed in huge chunks of code. The developers will spend their whole lives trying to explain to the non-programmers how the product works. So, the simplest way to explain and document the whole process is through a diagram. This is where StarUML comes in. The development team can show a diagrammatic representation of the software product and this can reduce time and documentation will be easy.

**OOSE (Object-Oriented Software Engineering):**

In object-oriented software engineering, the software developer identifies and organizes the application in terms of object-oriented concepts, prior to their final representation in any specific programming language or software tools.

**Difference between structured and object-oriented analysis**

|  |  |
| --- | --- |
| **Structured Analysis** | **Object-Oriented Analysis** |
| The main focus is on the process and procedures of the system. | The main focus is on data structure and real-world objects that are important. |
| It uses System Development Life Cycle (SDLC) methodology | It uses Incremental or Iterative methodology |
| It is suitable for well-defined projects with stable user requirements. | It is suitable for large projects with changing user requirements. |
| Risk while using this analysis technique is high and reusability is also low. | Risk while using this analysis technique is low and reusability is also high. |
| This technique is old and is not preferred usually. | This technique is new and is mostly preferred. |

**B.2 Problem Statement of the Case Study Selected**

**B.3 Conclusion**

*(Students must write the conclusion as per the problem definition noted in section B.2 )*

Thus we successfully completed the case study.

**B.4 Question of curiosity:**

1. Justify the statement, *“ User requirements must be understood well before beginning the software development”*

Before starting to create a software, it's important to understand what the users want and need from it. If we don't know this, it's unlikely that the software will be useful or successful. So, we should spend time to understand user requirements before beginning the development process.

1. What is SRS? Download sample IEEE SRS format and prepare the relevant part of the same for this experiment

Purpose, Scope, Functional and Non-functional Requirements

SRS stands for Software Requirements Specification. It is a detailed document that describes the requirements and specifications for a software system. The purpose of an SRS is to provide a clear and complete understanding of what the software should do and how it should perform. The scope of an SRS defines the boundaries of the software system, including its functions, features, and limitations. Functional requirements describe what the software should do, while non-functional requirements specify how well it should perform in terms of factors such as reliability, usability, and performance. IEEE provides a standard format for SRS, which includes sections for the purpose, scope, functional and non-functional requirements, among others.

1. What is feasibility study? When is the feasibility study done?

A feasibility study is an assessment that is done to determine if a project or idea is feasible or viable. It is typically conducted in the early stages of a project, before any significant resources are committed. The study evaluates the project's technical, economic, and operational aspects to determine if it can be successfully completed within the given constraints of time, budget, and resources. The purpose of the feasibility study is to provide decision-makers with the information they need to determine if the project should move forward or be abandoned.

1. Differentiate between functional requirements and nonfunctional requirements

Functional requirements describe what the software system should do and how it should behave in response to specific inputs or events. They define the system's functionality, features, and capabilities. Nonfunctional requirements, on the other hand, describe how the software system should perform in terms of qualities such as usability, reliability, performance, security, and maintainability. They define the system's overall quality attributes and constraints that must be satisfied for the software to be successful. While functional requirements focus on the system's behavior, nonfunctional requirements focus on the system's quality characteristics.

**PART A**

**EXPERIMENT NO. 2**

**A.1 Aim: - Modeling UML Use Case diagrams and capturing Use Case scenarios.**

**A.2 Prerequisite**

**Concepts of Actor, Use Case and Relationships**

**A.3 Outcome**

After successful completion of this experiment students will be able to

Design solution using unified modeling language.

**A.4 Theory**

**Use case diagrams**

Use case diagrams belong to the category of behavioral diagram of UML diagrams. Use case diagrams aim to present a graphical overview of the functionality provided by the system. It consists of a set of actions (referred to as use cases) that the concerned system can perform, one or more actors, and dependencies among them.

**Actor**

An actor can be defined as an object or set of objects, external to the system, which interacts with the system to get some meaningful work done. Actors could be human, devices, or even other systems.

For example, consider the case where a customer *withdraws cash* from an ATM. Here, customer is a human actor.

Actors can be classified as below :

* **Primary actor**: They are principal users of the system, who fulfill their goal by availing some service from the system. For example, a customer uses an ATM to withdraw cash when he needs it. A customer is the primary actor here.
* **Supporting actor**: They render some kind of service to the system. "Bank representatives", who replenishes the stock of cash, is such an example. It may be noted that replenishing stock of cash in an ATM is not the prime functionality of an ATM.

In a use case diagram primary actors are usually drawn on the top left side of the diagram.

**Use Case**

A use case is simply a functionality provided by a system. Continuing with the example of the ATM, *withdraw cash* is a functionality that the ATM provides. Therefore, this is a use case. Other possible use cases includes, *check balance*, *change PIN*, and so on.

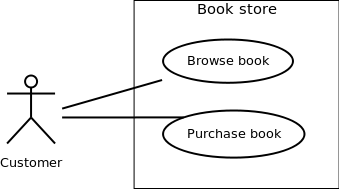
Use cases include both successful and unsuccessful scenarios of user interactions with the system. For example, authentication of a customer by the ATM would fail if he enters wrong PIN. In such case, an error message is displayed on the screen of the ATM.

**Subject**

Subject is simply the system under consideration. Use cases apply to a subject. For example, an ATM is a subject, having multiple use cases, and multiple actors interact with it. However, one should be careful of external systems interacting with the subject as actors.

**Graphical Representation**

An actor is represented by a stick figure and name of the actor is written below it. A use case is depicted by an ellipse and name of the use case is written inside it. The subject is shown by drawing a rectangle. Label for the system could be put inside it. Use cases are drawn inside the rectangle, and actors are drawn outside the rectangle, as shown in below figure:



Association between Actors and Use Cases

A use case is triggered by an actor. Actors and use cases are connected through binary associations indicating that the two communicates through message passing.

An actor must be associated with at least one use case. Similarly, a given use case must be associated with at least one actor. Association among the actors are usually not shown. However, one can depict the class hierarchy among actors.

**Use Case Relationships**

Three types of relationships exist among use cases:

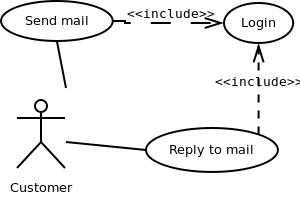
* Include relationship
* Extend relationship
* Use case generalization

**Include Relationship**

Include relationships are used to depict common behaviors that are shared by multiple use cases. This could be considered analogous to writing functions in a program in order to avoid repetition of writing the same code. Such a function would be called from different points within the program.

**Example**

For example, consider an email application. A user can send a new mail, reply to an email he has received, or forward an email. However, in each of these three cases, the user must be logged in to perform those actions. Thus, we could have a *login* use case, which is included bycompose *mail*, *reply*, and *forward email* use cases. The relationship is shown in below figure.



**Extend Relationship**

Use case extensions are used to depict any variation to an existing use case. They are used to specify the changes required when any assumption made by the existing use case becomes false.

**Example**

Let's consider an online bookstore. The system allows an authenticated user to buy selected book(s). While the order is being placed, the system also allows to specify any special shipping instructions, for example, call the customer before delivery. This *Shipping Instructions step* is optional, and not a part of the main *Place Order* use case. Below figure depicts such relationship.

<<extends>>

Place Order

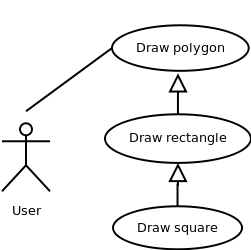
Shipping Instruction

**Generalization Relationship**

Generalization relationship are used to represent the inheritance between use cases. A derived use case specializes some functionality it has already inherited from the base use case.

**Example**

To illustrate this, consider a graphical application that allows users to draw polygons. We could have a use case *draw polygon*. Now, rectangle is a particular instance of polygon having four sides at right angles to each other. So, the use case *draw rectangle* inherits the properties of the use case *draw polygon* and overrides it's drawing method. This is an example of generalization relationship. Similarly, a generalization relationship exists between *draw rectangle* and *draw square* use cases.



**A.5 Task:**

For the problem statement, complete use case modelling in StarUML.

**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

*(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded on Blackboard LMS or emailed to the concerned Lab in charge Faculties at the end of practical; in case Blackboard is not accessible)*

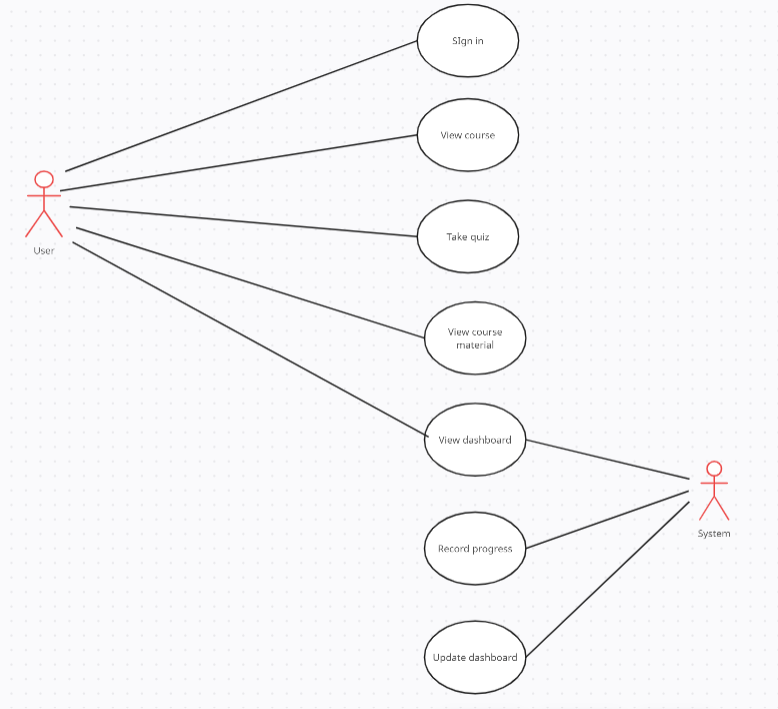
**B.1 Actors:**

*(Paste your document including as per format given)*

**B.2 Use cases:**

**B.3 Use Case diagrams:**

**B.4 Use Case Specifications:**

****

**B.5 Conclusion**

*(Students must write the conclusion as per the attainment of individual outcome listed above and project definition noted in section B.1 including “Define the System, Motivation, Scope of the System and Applications”)*

**We successfully identified the various use cases and how actors interact with them in Use Case Diagrams using UML.**

**B.5 Questions of Curiosity:**

Q1. What does a use case diagram represent?

1. A set of actions
2. Time sequence of statements executed
3. How to use a particular module
4. Don’t know

Answer: a

Q.2 Generalization relationship exists between two use cases when

1. A use case derives from a base use case.
2. A use case derives from a base use case and specializes some of its inherited functionality.
3. A use case includes functionality of some other use case.
4. No two use cases can be related.

Answer: b

**PART A**

**EXPERIMENT NO. 3**

**A.1 Aim: -** To draw the behavioral view diagram: Sequence diagram, Collaboration diagram

**A.2 Prerequisite**

Determine the desired flow of action and their interaction with each other

**A.3 Outcome**

After successful completion of this experiment students will be able to -

1. Better understanding of the interaction diagrams.
2. Get familiar with sequence & collaboration diagrams.
3. Practice drawing the interaction diagrams using StarUML

**A.4 Theory**

Interaction diagrams describe how groups of objects collaborate in some behavior. An interaction diagram typically captures the behavior of a single use case. Interaction diagrams do not capture the complete behavior, only typical scenarios.

Diagram is used to describe some type of interactions among the different elements in the model. Interaction is part of the **dynamic behavior** of the system – snapshot of running system at a particular moment. Sequence diagram emphasizes on time sequence of messages collaboration diagram emphasizes on the structural organization of the objects that send and receive messages.

For sequence diagram things to be identified:

* Objects taking part in the interaction – three types of objects – Entity, Control, Boundary objects
* Message flow among objects
* The sequence in which messages are flowing
* Object organization

**Sequence Diagram -**

Sequence diagrams are a graphical way to illustrate a scenario:

* They are called sequence diagrams because they show the sequence of message passing between objects.
* Another big advantage of these diagrams is that they show when the objects are created and when they are destructed. They also show whether messages are synchronous or asynchronous

**Collaboration Diagram -**

They are the same as sequence diagrams but without a time axis:

* Their message arrows are numbered to show the sequence of message sending.
* They are less complex and less descriptive than sequence diagrams.
* These diagrams are very useful during design because you can figure out how objects communicate with each other.

**A.5 Procedure/Algorithm**

**A.5.1 Task:**

Draw a sequence diagram for the case study.

* Identify objects – entity, control, boundary objects
* Identify messages between objects.

**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

*(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded on Blackboard LMS or emailed to the concerned Lab in charge Faculties at the end of practical; in case Blackboard is not accessible)*

**B.1 Objects:**

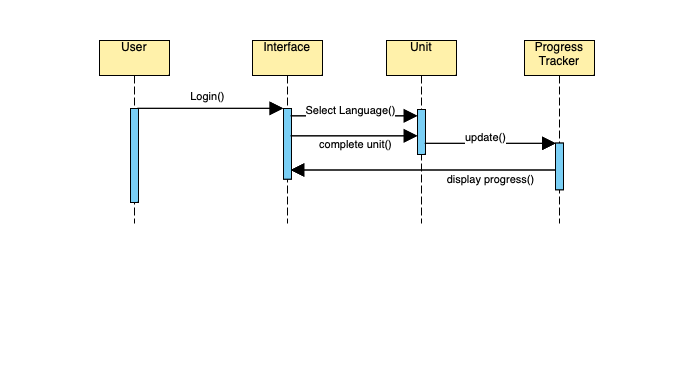
*(Paste your document including as per format given)*

**Entity objects:**

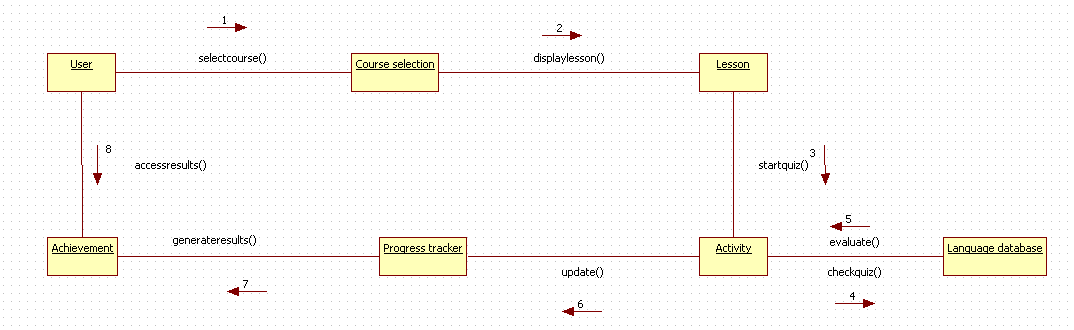
**Boundary objects:**

**Control objects:**

**B.2 Sequence diagram:**



**B.3 Collaboration diagram:**



**B.4 Conclusion**

*(Students must write the conclusion as per the attainment of individual outcome listed above and project definition noted in section B.1 including “Define the System, Motivation, Scope of the System and Applications”)*

**Thus we successfully drew and understood interaction diagrams for our project.**

**B.5 Questions of Curiosity:**

Q1. State the difference between entity, boundary and control objects.

Entity objects:

1. Represent the persistent data and behavior of the system.
2. Have long-term significance in the system and are not tied to specific use cases.
3. Interact with other entity objects to perform complex operations.

Boundary objects:

1. Represent the system's interactions with actors or external systems.
2. Handle input/output and user interface concerns.
3. Mediate communication between actors and the system.

Control objects:

1. Represent the logic that manages the interactions between entity and boundary objects.
2. Encapsulate complex use case-specific behaviors.
3. May coordinate multiple boundary and entity objects to achieve specific goals.

Q.2 State the difference between sequence and collaboration diagram.

1. Focus and purpose: Sequence diagrams focus on the chronological flow of messages and interactions between objects or components, while collaboration diagrams focus on the relationships and interactions between objects or components in a more abstract and static manner.
2. Direction of interaction: In sequence diagrams, the direction of interaction is usually top-down, with messages passing between objects or components in a sequential order. In collaboration diagrams, the direction of interaction is usually left-right or top-down, with objects or components arranged in a more abstract manner to show their relationships.
3. Level of detail: Sequence diagrams tend to be more detailed and specific, showing the exact messages and interactions between objects or components. Collaboration diagrams tend to be more abstract, showing the relationships between objects or components without necessarily specifying the exact messages or interactions.

Q.3 When looping is required in sequence diagram?

Looping is required in sequence diagrams when a particular sequence of interactions needs to be repeated a number of times. This is often the case when a system needs to perform a certain action multiple times, or when a process requires iteration. In such situations, the use of loops helps to simplify the diagram and make it more concise.

**PART A**

**EXPERIMENT NO. 4**

**A.1 Aim: - Modeling UML Class diagrams.**

**A.2 Prerequisite**

**Concepts of Actor, Use Case and Relationships, Sequence and collaboration diagram**

**A.3 Outcome**

After successful completion of this experiment students will be able to

Design solution using unified modeling language.

**A.4 Task:**

Complete Class Diagram in StarUML.

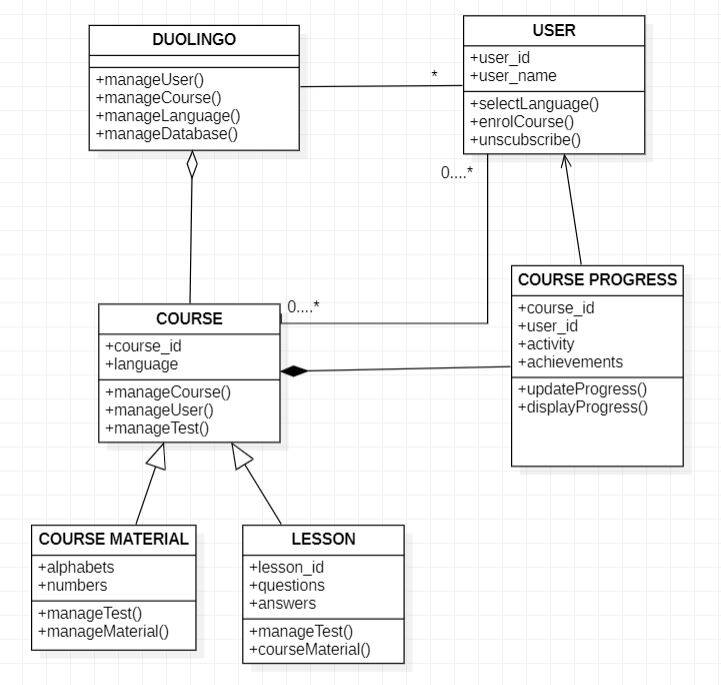
**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

*(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded on Blackboard LMS or emailed to the concerned Lab in charge Faculties at the end of practical; in case Blackboard is not accessible)*

**B.1**  **Design the class diagram**

*(Paste your class diagram including as per format given)*



**B.4 Conclusion**

*(Students must write the conclusion as per the attainment of individual outcome listed above and project definition noted in section B.1 including “Define the System, Motivation, Scope of the System and Applications”)*

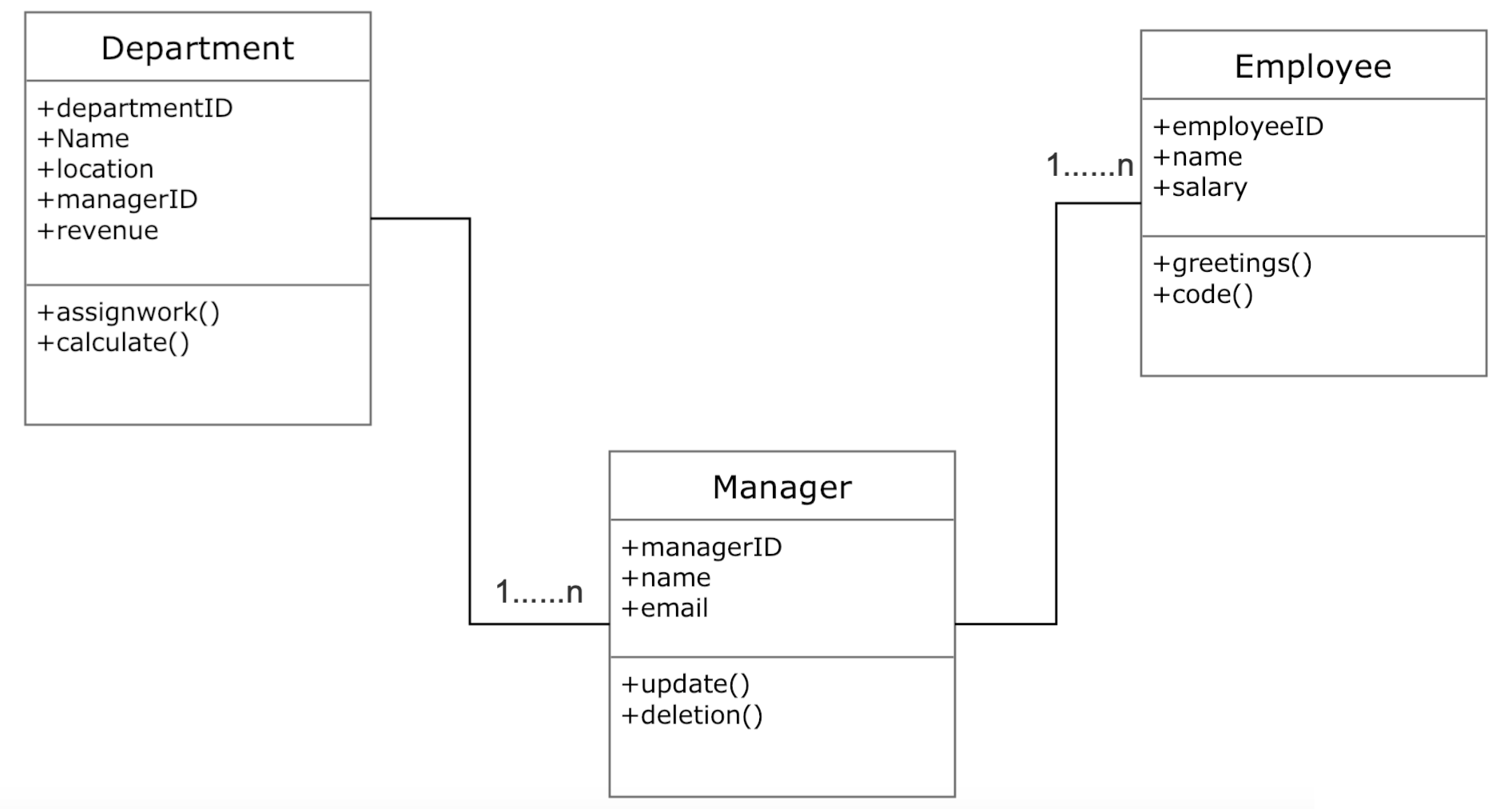
*Thus we understood class diagram which was already done in previous sem and drew the class diagram for our project.*

**B.5 Questions of Curiosity:**

Q1. A company consists of departments. Departments are distributed all over India. Each department has a manager. Department manager manages all the employees.

Design a class diagram, considering appropriately.

=>



**PART A**

**EXPERIMENT NO. 5**

**A.1 Aim: -** To draw the behavioral view diagram: Activity diagram

**A.2 Prerequisite**

Determine the desired flow of action and their interaction with each other

**A.3 Outcome**

After successful completion of this experiment students will be able to -

1. Better understanding of the interaction diagrams.
2. Get familiar with Activity diagram
3. Practice drawing the interaction diagrams using StarUML

**A.4 Theory**

Activity diagrams are flow charts that are used to show the workflow of a system.

They also:

* Represent the dynamics of the system.
* Show the flow of control from activity to activity in the system.
* Show what activities can be done in parallel, and any alternate paths through the flow.

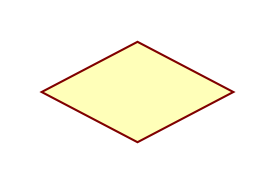
Activity diagrams may be created to represent the flow across use cases or they may be created to represent the flow within a particular use case. Later in the life cycle, activity diagrams may be created to show the workflow for an operation.

Activity diagram notations:

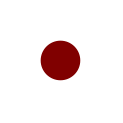
* Rounded rectangles represent activities



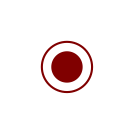
* Diamonds represent decisions



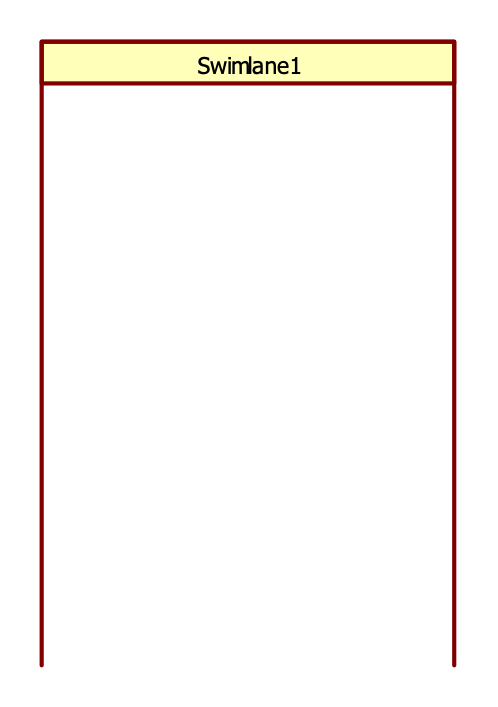
* A black circle represents the start (initial state) of the work-flow



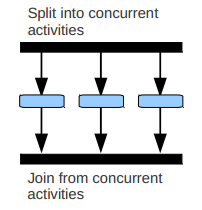
* An encircled black circle represents the end (final state).



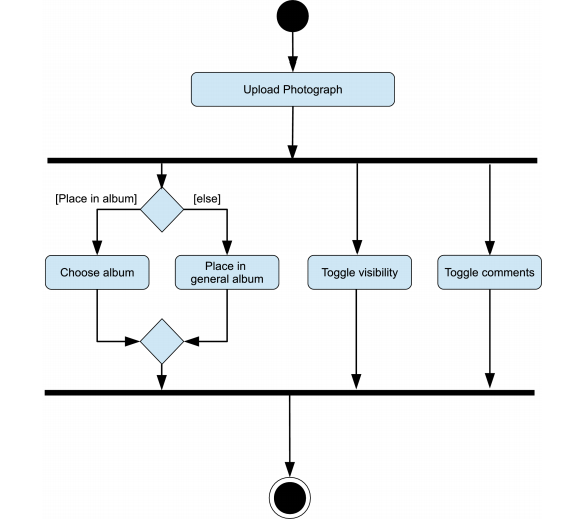
* Swimlane (vertical)
* Swimlane (horizontal): Swim lane- depicts which human organization is responsible for an activity. Organization – sales, finance, marketing, purchasing etc. Swim lane indicates that activity is performed by a person or persons within the organization.



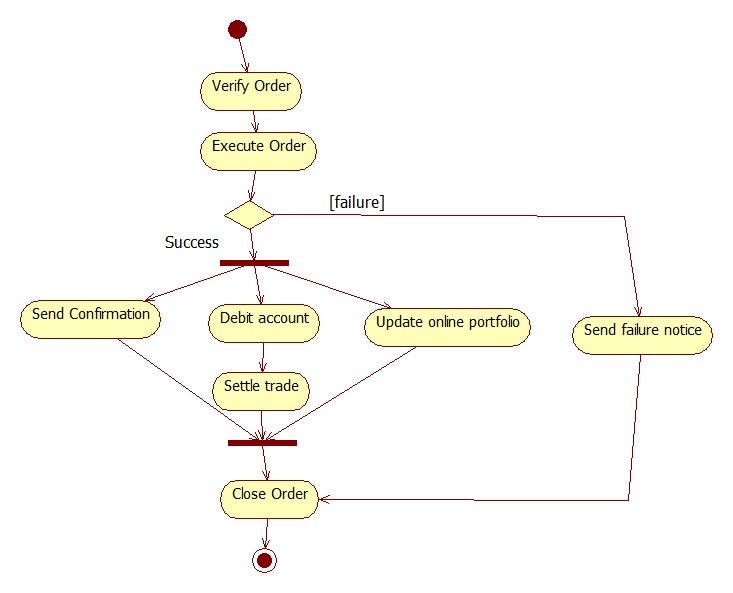
* Bars represent the start (split) or end (join) of concurrent activities



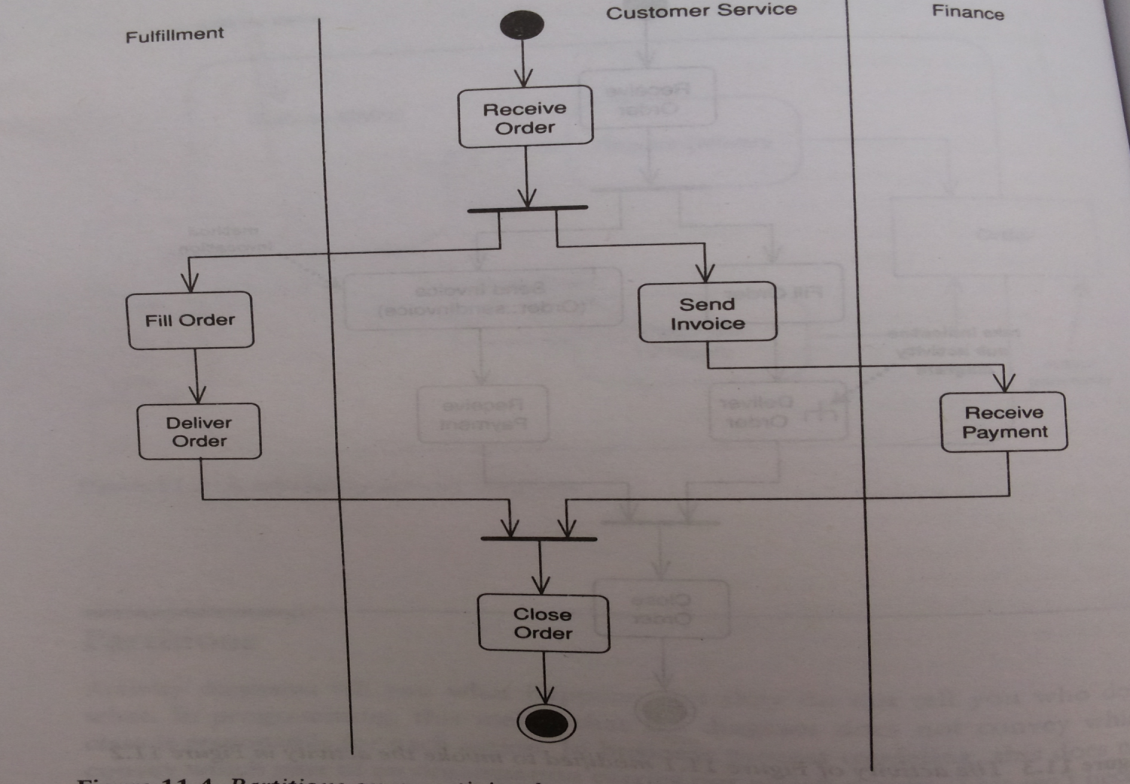
**Activity diagram for uploading photograph:**



**Activity diagram for stock trading processing:**

****

**Activity diagram using** **swimlane:**

****

**A.5. Task:**

Draw an activity diagram for the case study.

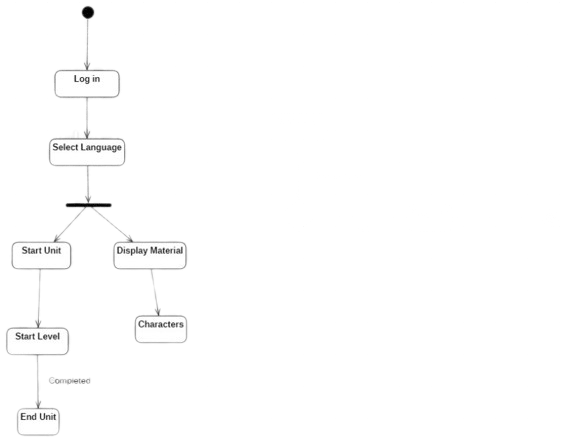
**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

*(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded on Blackboard LMS or emailed to the concerned Lab in charge Faculties at the end of practical; in case Blackboard is not accessible)*

**B.1 Activity diagram**

*(Paste your document including as per format given)*



**B.2 Conclusion**

*(Students must write the conclusion as per the attainment of individual outcome listed above and project definition noted in section B.1 including “Define the System, Motivation, Scope of the System and Applications”)*

*Thus we drew and understood activity diagram and how it is different than state chart.*

**B.5 Questions of Curiosity:**

Q1. What is the primary purpose of activity diagram?

The primary purpose of an activity diagram is to depict the flow of activities in a system or process. It provides a visual representation of the steps, decisions, and actions involved in a process, and helps to identify potential bottlenecks and areas for improvement. Activity diagrams can be used to model business processes, software applications, and even physical systems.

Activity diagrams are particularly useful for modeling complex processes or systems with many interacting components. They allow for the visualization of the entire process and provide a clear understanding of how each step contributes to the overall outcome. This makes it easier to identify inefficiencies or errors in the process and to design improvements.

In addition, activity diagrams can be used to communicate the process to others, such as stakeholders or development teams. They provide a clear and concise way of describing the steps involved in a process, and can be easily understood by people with different backgrounds and levels of expertise.

Q.2 State the difference between branches and fork and join in activity diagram.

Branches:

1. Branches are used to represent conditional logic in an activity diagram.
2. It allows the process to take different paths based on certain conditions.
3. It can represent a decision point in the process flow.

Fork and Join:

1. Fork and join are used to represent concurrent processing in an activity diagram.
2. It allows multiple activities to be executed in parallel.
3. Fork is used to split a process flow into multiple concurrent flows, and Join is used to combine them again.

**PART A**

**EXPERIMENT NO. 6**

**A.1 Aim: -** To draw the State Diagram

**A.2 Prerequisite**

Determine the State Diagram for the case study.

**A.3 Outcome**

After successful completion of this experiment students will be able to -

Draw the state diagrams using StarUML

**A.4 Procedure/Algorithm**

**A.4.1 Task:**

Draw a state diagram for the case study.

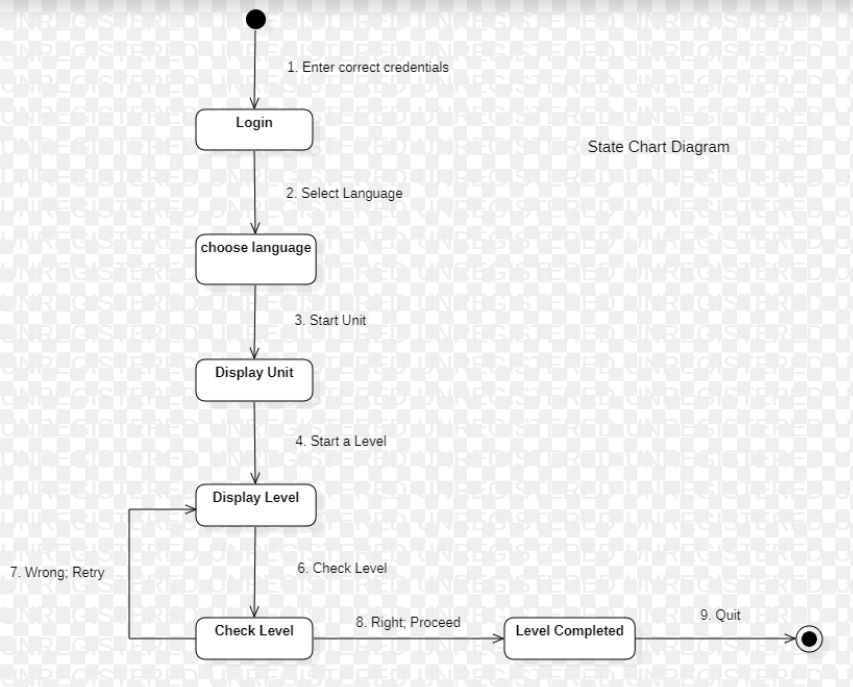
**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

*(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded on Blackboard LMS or emailed to the concerned Lab in charge Faculties at the end of practical; in case Blackboard is not accessible)*

**B.1 State Diagram**

*(Paste your document including as per format given)*



**B.4 Conclusion**

*(Students must write the conclusion as per the attainment of individual outcome listed above and project definition noted in section B.1 including “Define the System, Motivation, Scope of the System and Applications”)*

Thus we drew and understood state chart diagram as well as how it suddenly doesn’t have decision state.

**PART A**

**EXPERIMENT NO. 7**

**A.1 Aim: -** To draw Component Diagram

**A.2 Outcome**

After successful completion of this experiment students will be able to –

Practice drawing the component and deployment diagram using StarUML

**A.3 Theory**

This exercise focuses on component diagram, which depict the implementation of a system.

Component modeling is a specialized type of structural modeling concerned with modeling the implementation of a system. Using the UML, you can communicate the implementation of a system using component diagrams. You usually apply component modeling during design activities to determine how implementation activities will build the system; that is, to determine the elements of the system on which implementation activities will focus. Component modeling typically starts after the design of the system is fairly complete, as determined by your system development process.

1. **Component**

A component is a part of the system that exists when the system is executing. For example, the project management system may be decomposed into the following components:

*A user interface component*

Responsible for providing a user interface through which users may interact with the system

*A business-processing component*

Responsible for implementing business functionality, including all the project management functionality provided by the project management system

*A data component*

For implementing data storage functionality

*A security component*

Provides various forms of security functionality to the business-processing and data components, including user authentication and verifying user privileges when accessing data

You can use the UML to talk about classes of components as well as specific components of a class. When speaking of a class of components, it's customary to use the terms component or *component class*. Thus, while you might think of a component as a specific thing, in the UML, a component really represents a class of things. When speaking of a specific component of a class, use the term *component instance*.

A component exists during execution time and requires a resource on which to execute,. In the UML, a component is shown as a rectangle with two small rectangles protruding from its side. The rectangle is labelled with the name of the component class.

Figure 1 shows various components associated with the project management system, including user interface, business-processing, data, and security components.

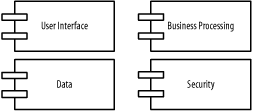


Figure 1- Components of the project management system

A component instance is a specific component. For example, specific components of the project management system include:

*A web user interface component instance*

Allows users to access the project management system via the Web

*A client/server user interface component instance*

Allows users to access the project management system in a client/server environment

*A local data component instance*

Stores project management data for a specific user or group of users

*An enterprise data component instance*

Stores project management data for a complete organization

A component instance is shown similar to a component class, but is labelled with the component instance name followed by a colon followed by the component class name, with all parts of the name fully underlined. Both names are optional, and the colon is present only if the component class name is specified.

Figure 2 shows various component instances of the component classes in Figure 1, including two user interface component instances, named Web and Client Server, two data component instances, named Local Data and Enterprise Data, a nameless business processing component instance, and a nameless security component instance.

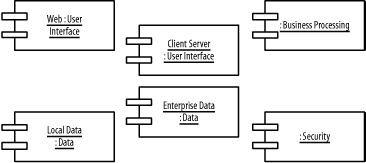


Figure 2- Component instances in the project management system

1. **Nodes**

A *node* is a resource that is available during execution time. Traditionally, nodes refer to computers on a network, but in the UML a node may be a computer, printer, server, Internet, or any other kind of resource available to components. For example, the project management system may be deployed on the following nodes:

*A desktop client*

On which the user interface component executes

*A printer*

Which the project management system uses to print reports

*A business-processing server*

On which the business-processing component executes

*A database server*

On which the data component executes and where project-related information is stored.

Nodes follow the type-instance dichotomy and applied to classes and objects. You can use the UML to talk about classes of nodes, as well as specific nodes of a class. When speaking of a class of nodes, it's customary to use the terms node or *node class*. Thus, while you might think of a node as a specific thing, in the UML, a node really represents a class of nodes. When speaking of a specific component of a class, use the term *node instance*.

A node is available during execution time and is a resource on which components may execute. In the UML, a node is shown as a three-dimensional rectangle labelled with the node's name.

Figure 3 shows various nodes associated with the project management system, including a desktop client, business-processing server, database server, and printer node.

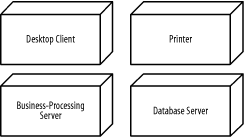


Figure 3- Nodes used by the project management system

A node instance is a specific node. For example, specific nodes used by the project management system include:

*A desktop client node instance*

Used by Jonathan to access the project management system

*A desktop client node instance*

Used by Andy to access the project management system

*A group business-processing server node instance*

Used by a group of users to manage projects

*An enterprise business-processing server node instance*

Used by a complete organization to manage projects

A node instance is shown similarly to a node class but labelled with the node instance name followed by a colon followed by the node class name, all fully underlined. Both names are optional, and the colon is present only if the node class name is specified.

Figure 4 shows various node instances of the node classes in Figure 3, including two desktop client node instances, named Jonathan's Computer and Andy's Computer, two business-processing node instances, named Group Server and Enterprise Server, a printer node instance, named Group Printer, and a database server node instance.

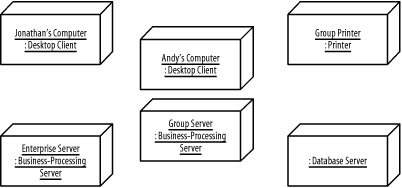


Figure 4- Node instances

1. **Dependencies**

Figure 1 shows components associated with the project management system, and Figure 3 shows nodes associated with the project management system, but how are components related to undifferentiated and differentiated classes, packages, subsystems, and to other components and nodes? Specialized types of dependencies called reside, use, and deploy dependencies address these questions. The next few sections discuss these specialized types of dependencies.

**3.1 Reside Dependencies**

A *reside* dependency from a component to any UML element indicates that the component is a client of the element, which is itself considered a supplier, and that the element resides in the component. The element may be an undifferentiated or differentiated class, package, or subsystem. An element may reside in any number of components, and a component may have any number of elements that reside in it.

A reside dependency is shown as a dashed arrow from a client component to a supplier element marked with the *reside* keyword.

Figure 5 shows that the User Interface and Utility packages reside in the User Interface component. Because the User Interface package depends on the Utility package, the User Interface and Utility packages must reside in the same component; otherwise, the User Interface package would not be able to use the Utility package.

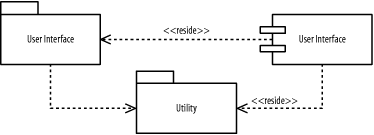


Figure 5. Reside dependencies for packages

Figure 6 shows that the Business Processing subsystem and Utility package reside in the Business Processing component. Because the Business Processing subsystem provides the Business Processing interface, the Business Processing component also provides the interface. Again, because the Business Processing subsystem depends on the Utility package, the Business Processing subsystem and Utility package must reside in the same component; otherwise, the Business Processing subsystem would not be able to use the Utility package. Remember, it's perfectly fine for an element to reside in more than one component. For example, the Utility package resides in both the User Interface and Business Processing components, and, as you will soon see, in the Data component.

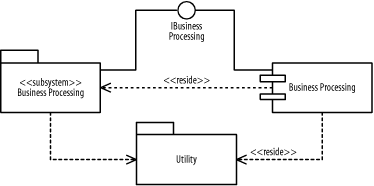


Figure 6. Reside dependencies for subsystems

Alternatively, an element that resides inside a component may be shown nested inside the component. Figure 7 shows that the Data subsystem and Utility package reside in the Data component. The Data subsystem is drawn inside the Data component, while thereside dependency to Utility is still drawn in the same manner as in Figures Figure 5 and Figure 6.

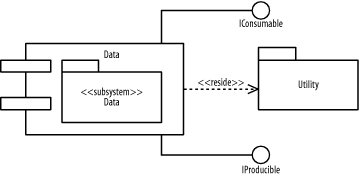


Figure 7. Reside dependencies using nesting

Notice that the Utility package resides in all the components in Figures Figure 5, Figure 6, and Figure 7, because each component described in those figures has a package that uses the Utility package.

**3.2 Use Dependencies**

A *use*dependency from a client component to a supplier component indicates that the client component uses or depends on the supplier component. A use dependency from a client component to a supplier component's interface indicates that the client component uses or depends on the interface provided by the supplier component. A use dependency is shown as a dashed arrow from a client component to a supplier component or a supplier component's interface. The dependency may be marked with the use keyword; however, the keyword is often omitted because this is the default, and the meaning is evident from how the dependency is used.

Figure 8 shows how the various components of the project management system are related:

The User Interface component-

Uses the Security component and the IBusinessProcessing interface provided by the Business Processing component

The Business Processing component-

Uses the Security component and the IProducible and IConsumable interfaces provided by the Data component

The Data component-

Uses the Security component

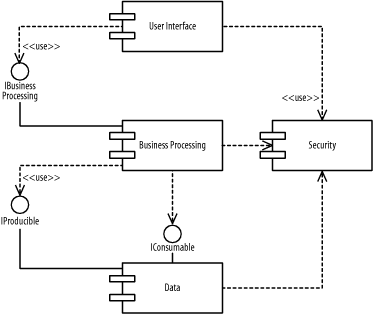


Figure 8- Use dependencies

**3.3 Deploy Dependencies**

A *deploy*dependency from a client component to a supplier node indicates that the client component is deployed on the supplier node.

A deploy dependency is shown as a dashed arrow from a client component to a supplier node marked with the *deploy* keyword.

Figure 9 shows that the User Interface component is deployed on the Desktop Client node.

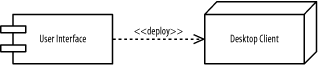


Figure 9- Deploy dependencies

Figure 10 shows that the *Business Processing* component is deployed on the *Business-Processing Server*node.

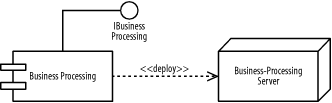


Figure 10- Deploy dependencies for a subsystem

Alternatively, a component that is deployed on a node may be shown nested inside the node. Figure 11 shows that the Data component is deployed on the *Database Server* node.

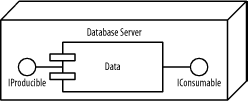


Figure 11- Deploy dependencies using nesting

**A.4 Procedure/Algorithm**

**A.4.1 Task:**

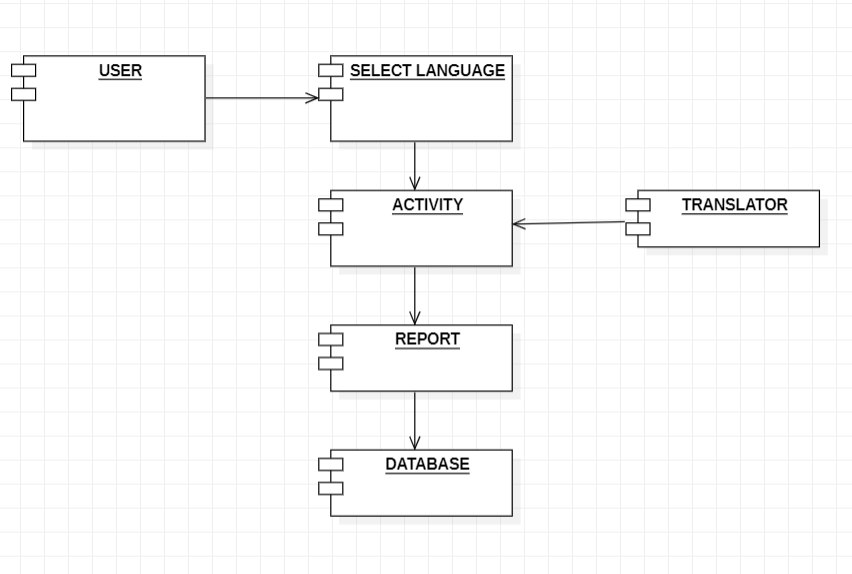
For the case study given on black board complete component and deployment model.

**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

*(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded on Blackboard LMS or emailed to the concerned Lab in charge Faculties at the end of practical; in case Blackboard is not accessible)*

**Diagram:**



**B.4 Conclusion**

*(Students must write the conclusion as per the attainment of individual outcome listed above and project definition noted in section B.1 including “Define the System, Motivation, Scope of the System and Applications”)*

*Thus we completed component diagram.*

**Part A**

**EXPERIMENT – Package Diagram**

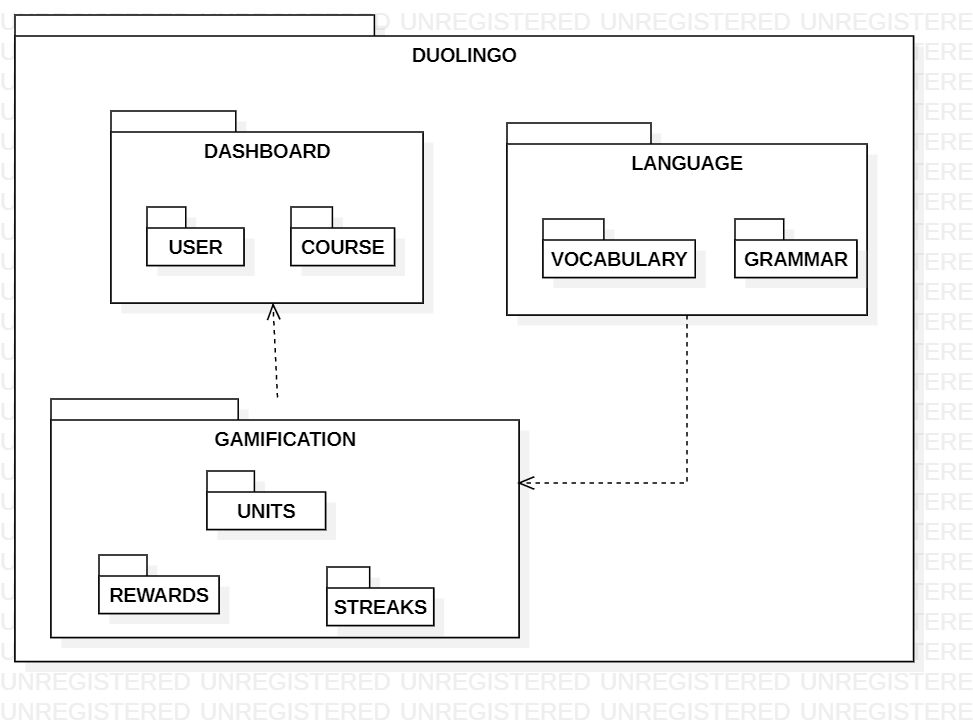
**AIM: -** To draw Package Diagram using StarUML

**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

*(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded on Blackboard LMS or emailed to the concerned Lab in charge Faculties at the end of practical; in case Blackboard is not accessible)*

**Diagram:**



**Conclusion:** Thus, we successfully understood and made package diagram using StarUML for our project.

**PART A**

**EXPERIMENT NO. 8**

**A.1** **Aim: -** To draw Deployment Diagram

**A.2 Outcome**

After successful completion of this experiment students will be able to -

1. Practice drawing the deployment diagram using StarUML

**A.3** **Theory**

This exercise focuses on component diagram, which depict the implementation of a system.

Component modeling is a specialized type of structural modeling concerned with modeling the implementation of a system. Using the UML, you can communicate the implementation of a system using component diagrams. You usually apply component modeling during design activities to determine how implementation activities will build the system; that is, to determine the elements of the system on which implementation activities will focus. Component modeling typically starts after the design of the system is fairly complete, as determined by your system development process.

1. **Component**

A component is a part of the system that exists when the system is executing. For example, the project management system may be decomposed into the following components:

*A user interface component*

Responsible for providing a user interface through which users may interact with the system

*A business-processing component*

Responsible for implementing business functionality, including all the project management functionality provided by the project management system

*A data component*

For implementing data storage functionality

*A security component*

Provides various forms of security functionality to the business-processing and data components, including user authentication and verifying user privileges when accessing data

You can use the UML to talk about classes of components as well as specific components of a class. When speaking of a class of components, it's customary to use the terms component or *component class*. Thus, while you might think of a component as a specific thing, in the UML, a component really represents a class of things. When speaking of a specific component of a class, use the term *component instance*.

A component exists during execution time and requires a resource on which to execute,. In the UML, a component is shown as a rectangle with two small rectangles protruding from its side. The rectangle is labelled with the name of the component class.

Figure 1 shows various components associated with the project management system, including user interface, business-processing, data, and security components.

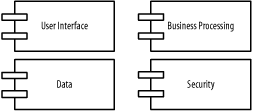


Figure 1- Components of the project management system

A component instance is a specific component. For example, specific components of the project management system include:

*A web user interface component instance*

Allows users to access the project management system via the Web

*A client/server user interface component instance*

Allows users to access the project management system in a client/server environment

*A local data component instance*

Stores project management data for a specific user or group of users

*An enterprise data component instance*

Stores project management data for a complete organization

A component instance is shown similar to a component class, but is labelled with the component instance name followed by a colon followed by the component class name, with all parts of the name fully underlined. Both names are optional, and the colon is present only if the component class name is specified.

Figure 2 shows various component instances of the component classes in Figure 1, including two user interface component instances, named Web and Client Server, two data component instances, named Local Data and Enterprise Data, a nameless business processing component instance, and a nameless security component instance.

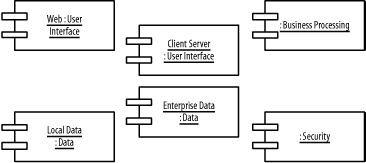


Figure 2- Component instances in the project management system

1. **Nodes**

A *node* is a resource that is available during execution time. Traditionally, nodes refer to computers on a network, but in the UML a node may be a computer, printer, server, Internet, or any other kind of resource available to components. For example, the project management system may be deployed on the following nodes:

*A desktop client*

On which the user interface component executes

*A printer*

Which the project management system uses to print reports

*A business-processing server*

On which the business-processing component executes

*A database server*

On which the data component executes and where project-related information is stored.

Nodes follow the type-instance dichotomy and applied to classes and objects. You can use the UML to talk about classes of nodes, as well as specific nodes of a class. When speaking of a class of nodes, it's customary to use the terms node or *node class*. Thus, while you might think of a node as a specific thing, in the UML, a node really represents a class of nodes. When speaking of a specific component of a class, use the term *node instance*.

A node is available during execution time and is a resource on which components may execute. In the UML, a node is shown as a three-dimensional rectangle labelled with the node's name.

Figure 3 shows various nodes associated with the project management system, including a desktop client, business-processing server, database server, and printer node.

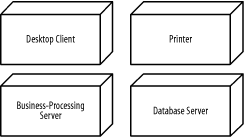


Figure 3- Nodes used by the project management system

A node instance is a specific node. For example, specific nodes used by the project management system include:

*A desktop client node instance*

Used by Jonathan to access the project management system

*A desktop client node instance*

Used by Andy to access the project management system

*A group business-processing server node instance*

Used by a group of users to manage projects

*An enterprise business-processing server node instance*

Used by a complete organization to manage projects

A node instance is shown similarly to a node class but labelled with the node instance name followed by a colon followed by the node class name, all fully underlined. Both names are optional, and the colon is present only if the node class name is specified.

Figure 4 shows various node instances of the node classes in Figure 3, including two desktop client node instances, named Jonathan's Computer and Andy's Computer, two business-processing node instances, named Group Server and Enterprise Server, a printer node instance, named Group Printer, and a database server node instance.

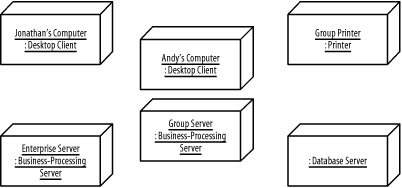


Figure 4- Node instances

1. **Dependencies**

Figure 1 shows components associated with the project management system, and Figure 3 shows nodes associated with the project management system, but how are components related to undifferentiated and differentiated classes, packages, subsystems, and to other components and nodes? Specialized types of dependencies called reside, use, and deploy dependencies address these questions. The next few sections discuss these specialized types of dependencies.

**3.1 Reside Dependencies**

A *reside* dependency from a component to any UML element indicates that the component is a client of the element, which is itself considered a supplier, and that the element resides in the component. The element may be an undifferentiated or differentiated class, package, or subsystem. An element may reside in any number of components, and a component may have any number of elements that reside in it.

A reside dependency is shown as a dashed arrow from a client component to a supplier element marked with the *reside* keyword.

Figure 5 shows that the User Interface and Utility packages reside in the User Interface component. Because the User Interface package depends on the Utility package, the User Interface and Utility packages must reside in the same component; otherwise, the User Interface package would not be able to use the Utility package.

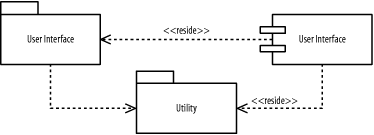


Figure 5. Reside dependencies for packages

Figure 6 shows that the Business Processing subsystem and Utility package reside in the Business Processing component. Because the Business Processing subsystem provides the IBusiness Processing interface, the Business Processing component also provides the interface. Again, because the Business Processing subsystem depends on the Utility package, the Business Processing subsystem and Utility package must reside in the same component; otherwise, the Business Processing subsystem would not be able to use the Utility package. Remember, it's perfectly fine for an element to reside in more than one component. For example, the Utility package resides in both the User Interface and Business Processing components, and, as you will soon see, in the Data component.

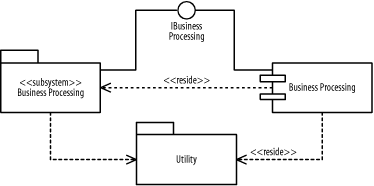


Figure 6. Reside dependencies for subsystems

Alternatively, an element that resides inside a component may be shown nested inside the component. Figure 7 shows that the Data subsystem and Utility package reside in the Data component. The Data subsystem is drawn inside the Data component, while thereside dependency to Utility is still drawn in the same manner as in Figures Figure 5 and Figure 6.

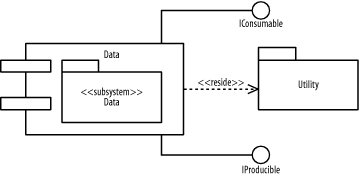


Figure 7. Reside dependencies using nesting

Notice that the Utility package resides in all the components in Figures Figure 5, Figure 6, and Figure 7, because each component described in those figures has a package that uses the Utility package.

**3.2 Use Dependencies**

A *use*dependency from a client component to a supplier component indicates that the client component uses or depends on the supplier component. A use dependency from a client component to a supplier component's interface indicates that the client component uses or depends on the interface provided by the supplier component. A use dependency is shown as a dashed arrow from a client component to a supplier component or a supplier component's interface. The dependency may be marked with the use keyword; however, the keyword is often omitted because this is the default, and the meaning is evident from how the dependency is used.

Figure 8 shows how the various components of the project management system are related:

The User Interface component-

Uses the Security component and the IBusinessProcessing interface provided by the Business Processing component

The Business Processing component-

Uses the Security component and the IProducible and IConsumable interfaces provided by the Data component

The Data component-

Uses the Security component

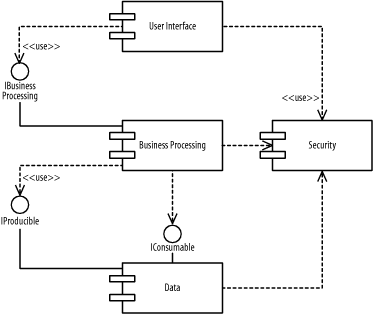


Figure 8- Use dependencies

**3.3 Deploy Dependencies**

A *deploy*dependency from a client component to a supplier node indicates that the client component is deployed on the supplier node.

A deploy dependency is shown as a dashed arrow from a client component to a supplier node marked with the *deploy* keyword.

Figure 9 shows that the User Interface component is deployed on the Desktop Client node.

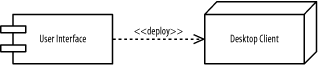


Figure 9- Deploy dependencies

Figure 10 shows that the *Business Processing* component is deployed on the *Business-Processing Server*node.

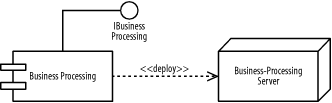


Figure 10- Deploy dependencies for a subsystem

Alternatively, a component that is deployed on a node may be shown nested inside the node. Figure 11 shows that the Data component is deployed on the *Database Server* node.

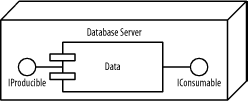


Figure 11- Deploy dependencies using nesting

**A.4 Procedure/Algorithm**

**A.4.1 Task:**

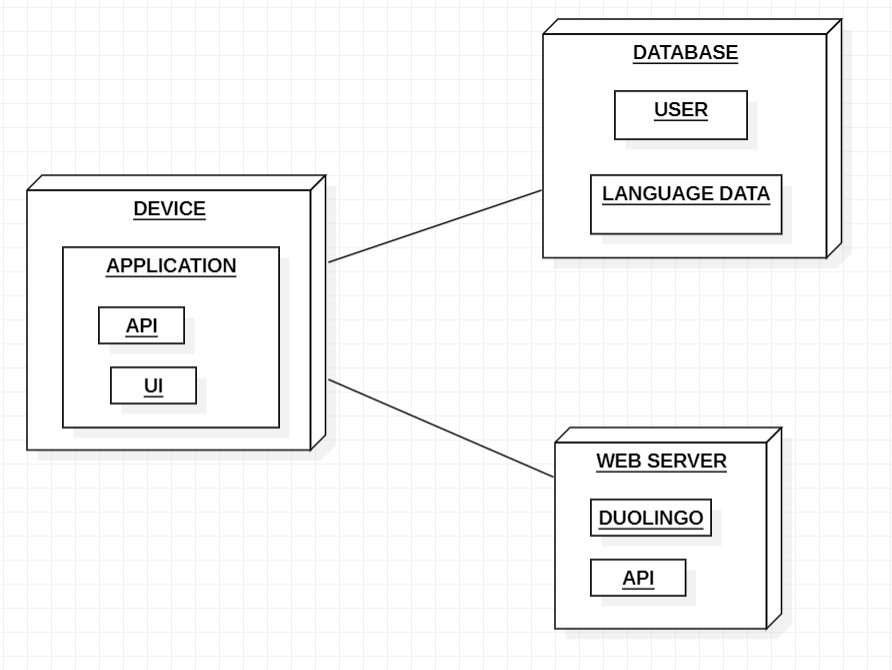
For the case study given on black board complete deployment model.

**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

*(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded on Blackboard LMS or emailed to the concerned Lab in charge Faculties at the end of practical; in case Blackboard is not accessible)*

**Diagram:**



**B.4 Conclusion**

*(Students must write the conclusion as per the attainment of individual outcome listed above and project definition noted in section B.1 including “Define the System, Motivation, Scope of the System and Applications”)*

*Thus we successfully drew Deployment Diagram on StarUML.*

**PART A**

**EXPERIMENT NO. 9**

**Experiment 9**

**A.1 Aim: Coding for any Two module of your project selected according to the designs which already designed and analyzed in EXP.1-6**

**A.2 Prerequisite:**

Use case diagram, class diagram, State diagram

**A.3 Outcome:**

Coding of a module with less errors

**A.4 Theory:**

The coding depends on individual’s project. Any programming language can be used according to student’s interest.

**Coding** : The objective of the coding phase is to transform the design of a system into code in a high-level language and then to unit test this code. Good software development organizations normally require their programmers to adhere to some well-defined and standard style of coding called coding standards.

Coding Standards- A coding standard gives a uniform appearance to the codes written by different engineers. It enhances code understanding. It encourages good programming practices.

**Coding Standards and Guideline:**

Limiting the use of global data type. Contents of the headers preceding codes for different modules naming conventions for global variables, local variables, and constant identifiers. Error return conventions and exception handling mechanisms Representative Coding Standards. Do not use a coding style that is too clever or too difficult to understand. Avoid obscure side effects. Do not use an identifier for multiple purposes. The code should be well-documented.

**Code Review:**

Code review for a model is carried out after the module is successfully compiled and all the syntax errors have been eliminated. Normally, two types of reviews are carried out on the code of a module.

**Code Walk Through:** To discover the algorithm and logical errors in the code.

**Code Inspection:** The aim of code inspection is to discover some common types of errors caused due to oversight and improper programming.

Software Documentation: Good documents are very useful and serves the following purposes. Good documents enhance understandability and maintainability of a software product. It helps the users in effectively using the system. Helps in effectively handling the manpower turnover problem. Helps the manager in effectively tracking the progress of the project.

Software Documentation classified into the following: Internal documentation: These are provided in the source code itself

External documentation: These are the supporting documents that usually accompany a software product

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

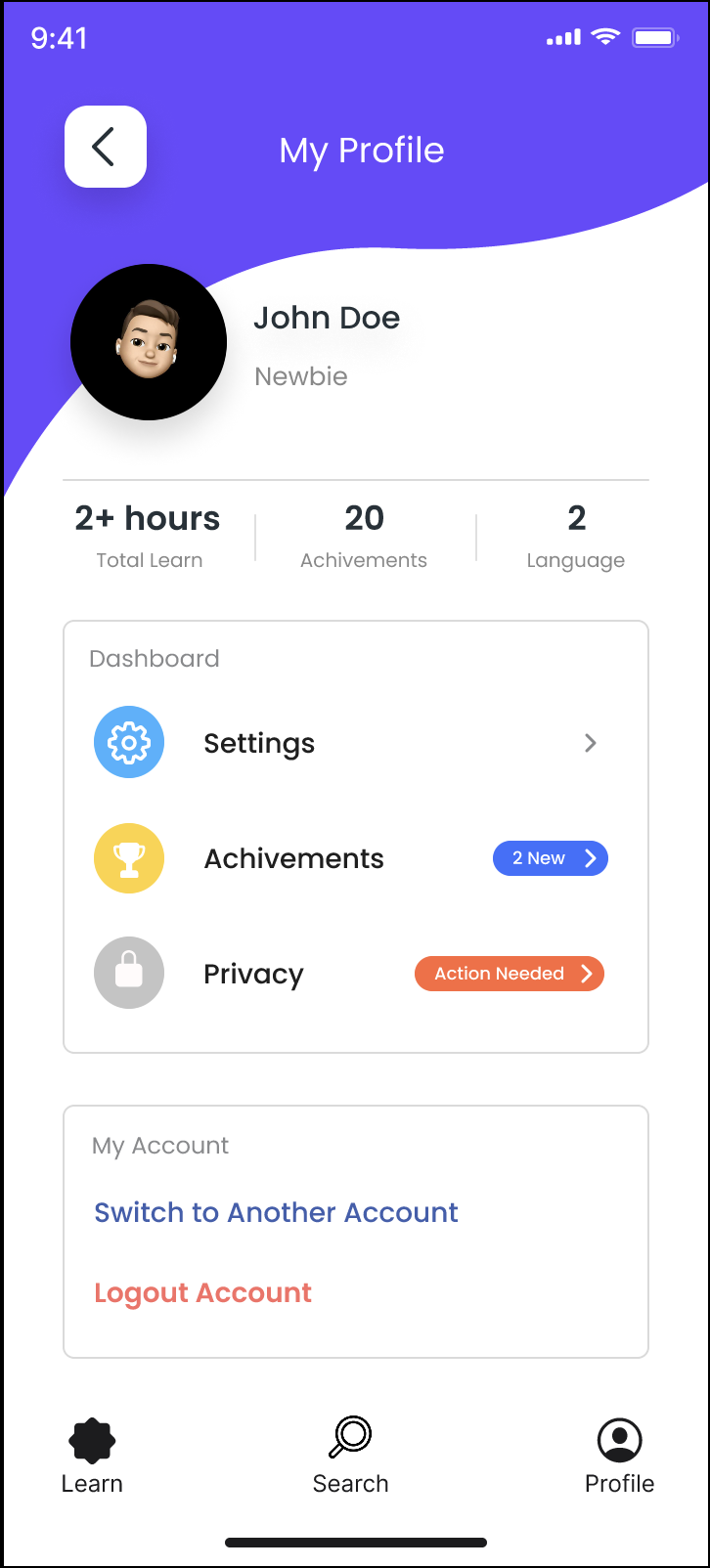
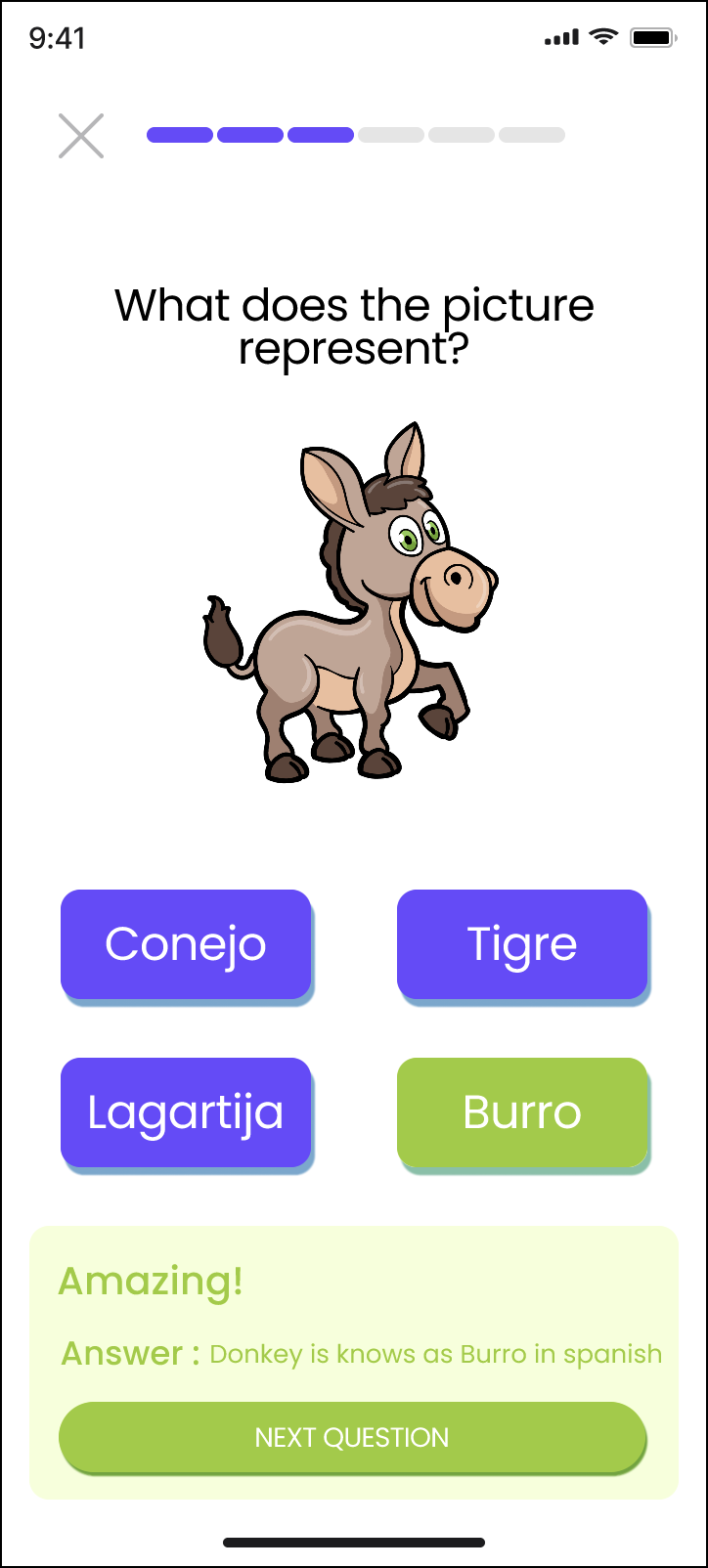
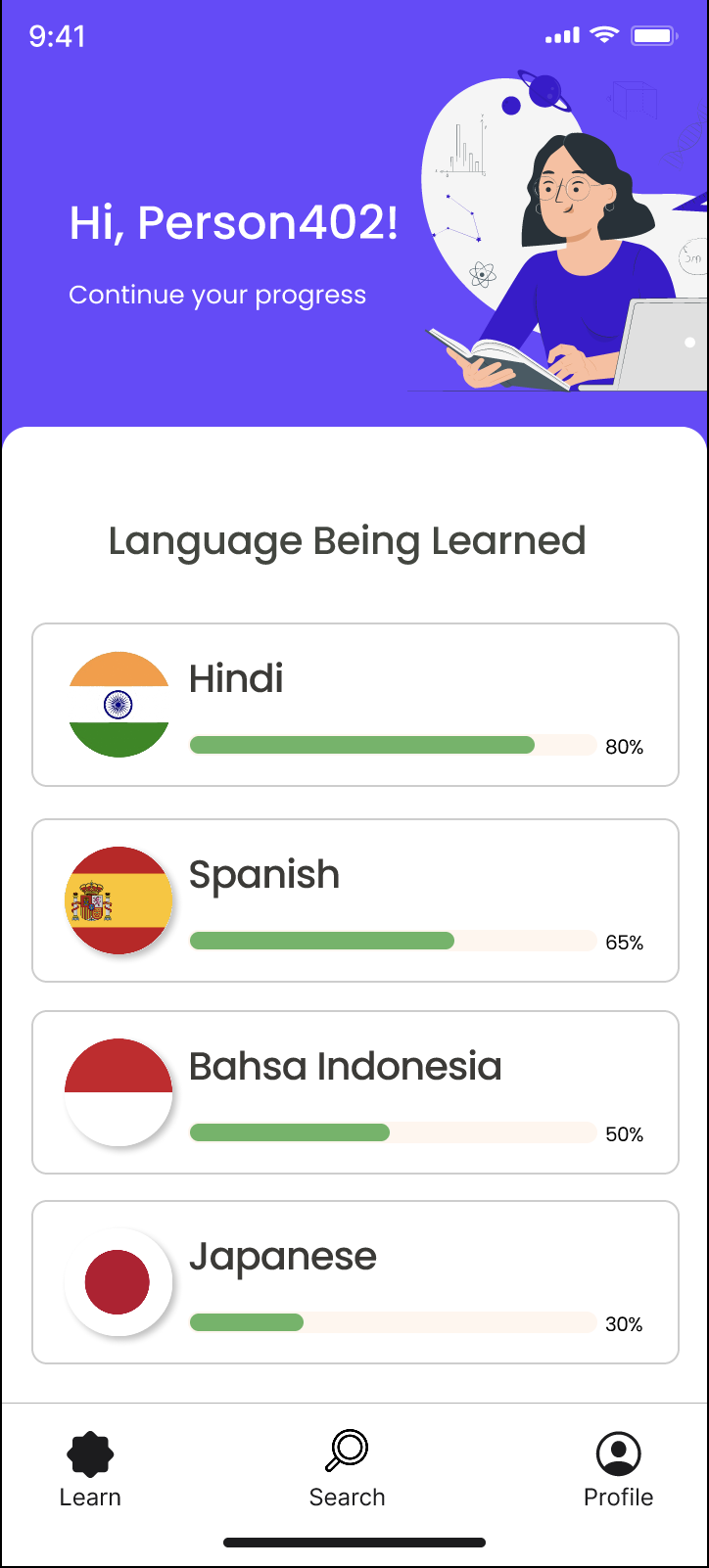
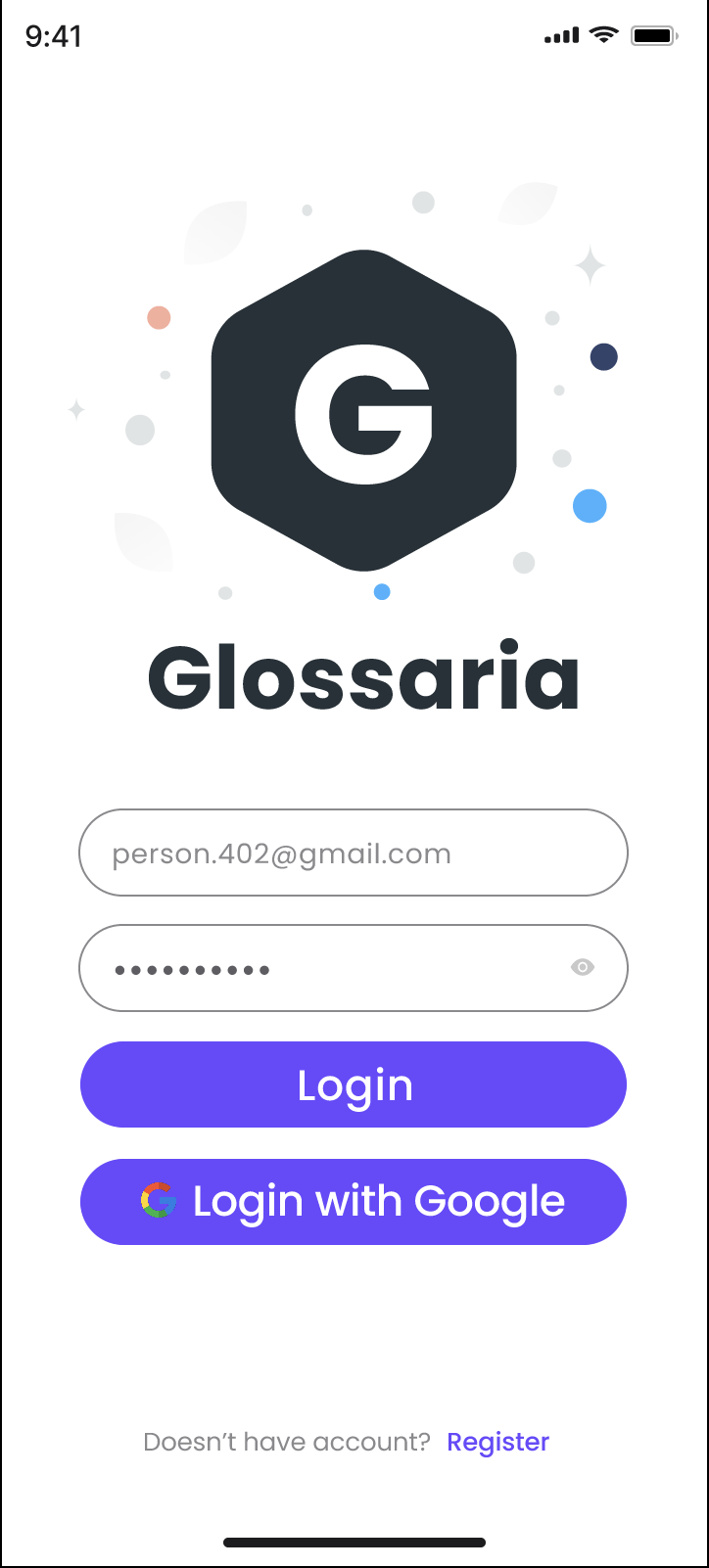
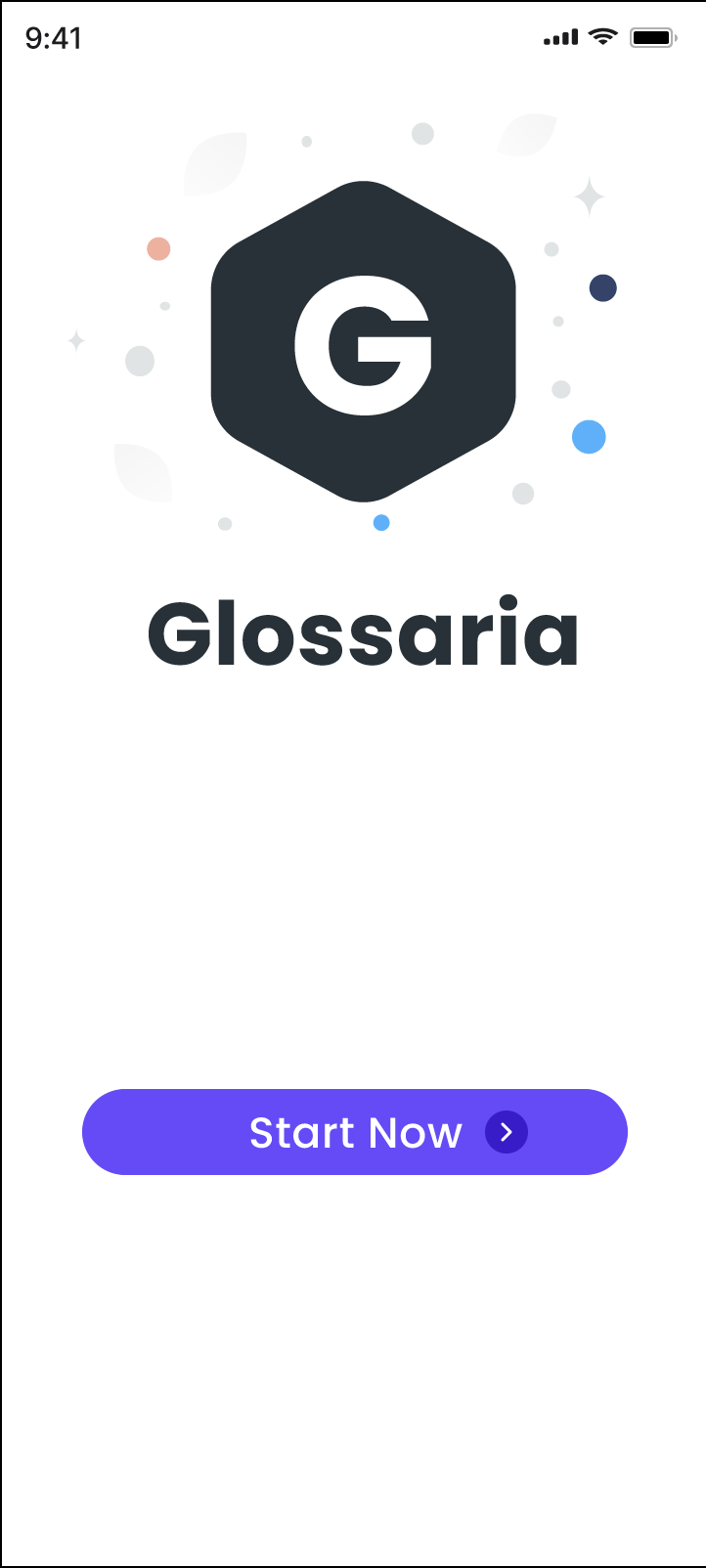
**PART B**

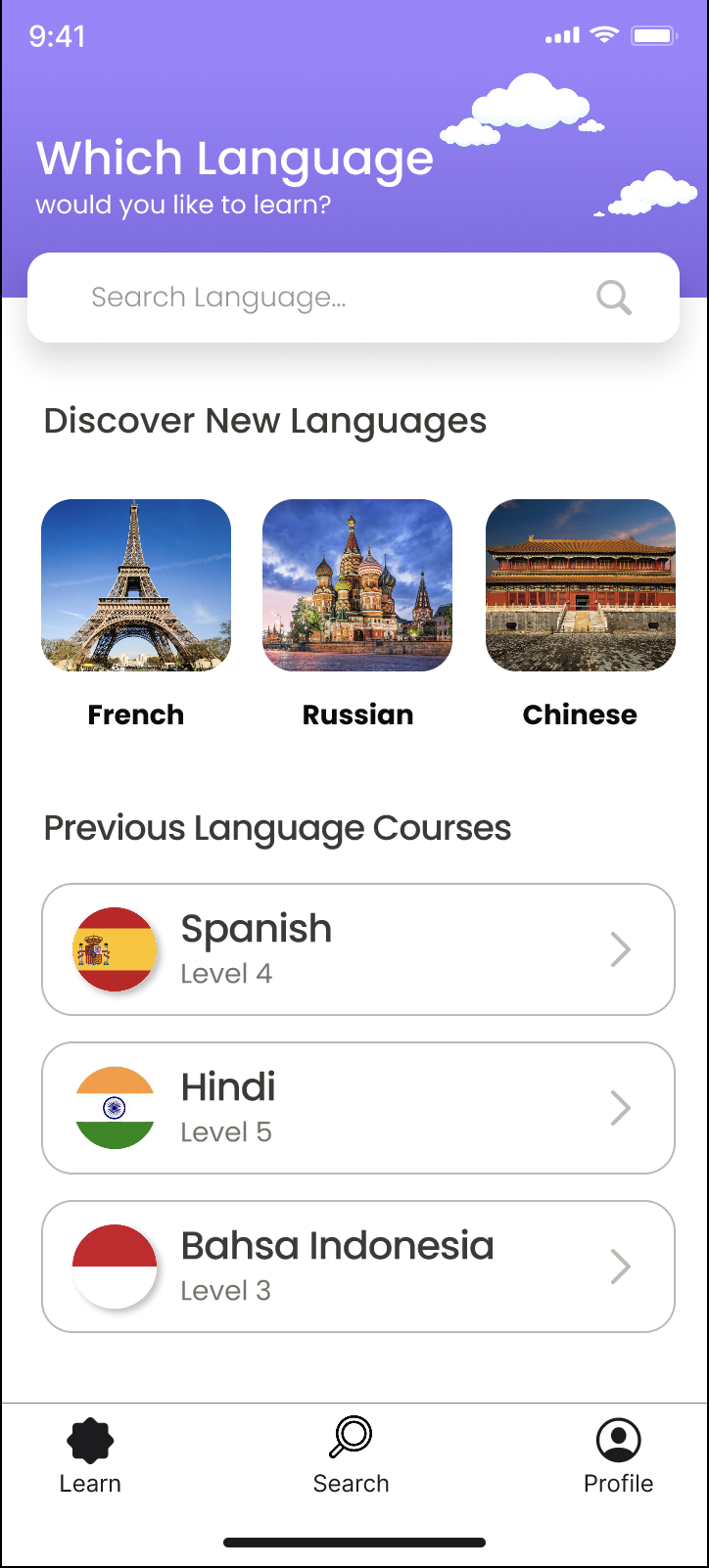
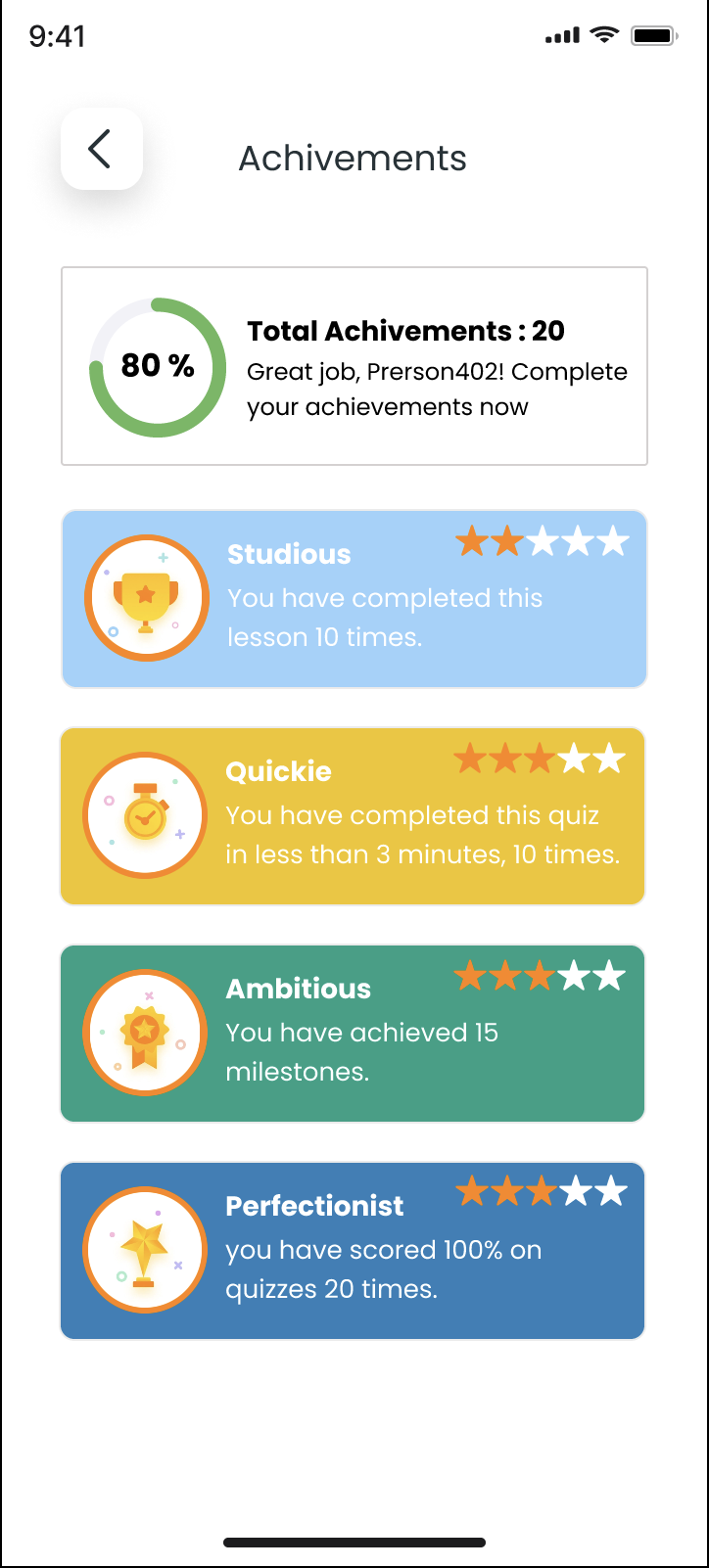
(PART B: TO BE COMPLETED BY STUDENTS)

*(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded on Blackboard LMS or emailed to the concerned Lab in charge Faculties at the end of practical; in case Blackboard is not accessible)*

**B.1 Task:**

Screen shot of the project developed with the proper explanation is required.





**B.1 Conclusion**

*(Students must write the conclusion as per the attainment of individual outcome listed above and project definition noted in section B.1 including “Define the System, Motivation, Scope of the System and Applications”)*

Thus we Successfully developed the proposed project after analyzing the diagrams we used in previous experiments.

**PART A**

**EXPERIMENT NO. 10**

**A.1 Aim: -** To Design Test Cases

**A.2 Outcome**

After successful completion of this experiment students will be able to design the test cases

Learn different testing methods and design test cases for their project

**A.3 Theory**

* Testing is the process of analyzing a system or system component to detect the differences between specified (required) and observed (existing) behavior.
* Activities involved in testing are:
  + Establish the test objectives
  + Design the test cases
  + Write the test cases
  + Test the test cases
  + Execute the tests
  + Evaluate the test results
  + Change the system
  + Do regression testing

1. Select what has to be tested
   * Analysis: Completeness of requirements
   * Design: Cohesion
   * Implementation: Source code
2. Decide how the testing is done
   * Review or code inspection
   * Proofs (Design by Contract)
   * Black-box, white box,
   * Select integration testing strategy (big bang, bottom up, top down, sandwich)
3. Develop test cases
   * A test case is a set of test data or situations that will be used to exercise the unit (class, subsystem, system) being tested or about the attribute being measured
4. Test plan
   1. Focuses on managerial aspects of testing
   2. Documents the scope, approach, resources and schedule of testing activities
   3. Requirements and the components to be tested are identified in this document
5. Test case specification
   1. Writing **effective test cases** is a skill and that can be achieved by some experience and in-depth study of the application on which test cases are being written

Test Case Specification Template:

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test case id** | **Test cases** | **Priority** | **Preconditions** | **Input test data** | **Steps to be executed** | **Expected results** | **Actual results** | **Pass/fail** | **Comments** |
| 1 | Test if user is able to login successfully. | A | User must be registered already | correct username,  correct password | 1)Enter input(correct )username and password on the respective fields 2)click submit/login | User must successfully login to the web page | (note down the results you have observed) |  |  |
| 2 | Test if unregistered users is not able to login to the site | A |  | incorrect username,incorrect password | 1)Enter input(incorrect )username and password on the respective fields 2)click submit/login | Proper error must be displayed and prompt to enter login again | (note down the results you have observed) |  |  |

**A.4 Task:**

For Selected Case Study write appropriate test cases and adopt necessary test comments to existing coding. Improve the quality of the coding which is performed in Experiment-9.

**PART B**

(PART B: TO BE COMPLETED BY STUDENTS)

*(Students must submit the soft copy as per the following segments within two hours of the practical. The soft copy must be uploaded on Blackboard LMS or emailed to the concerned Lab in charge Faculties at the end of practical; in case Blackboard is not accessible)*

**B.1 Task:**

For Selected Case Study write appropriate test cases and adopt necessary test comments to existing coding. Improve the quality of the coding which is performed in Experiment-9.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Test case id** | **Test cases** | **Priority** | **Preconditions** | **Input test data** | **Steps to be executed** | **Expected results** | **Actual results** | **Pass/**  **Fail** | **Comments** |
| 1 | Test if user is able to Register successfully | A | User must not have multiple accounts with same email id | New username, password, new email id | 1)Enter (new) input username and password and email id on the respective fields  2)click submit | User must successfully register as a new user. | User was able to successfully register as a new user. | Pass | - |
| 2 | Test if user is able to Login successfully | A | User must have registered earlier | Correct username, correct password | 1)Enter input (correct) username and password on the respective fields.  2)click submit/login | User must successfully login to their account | User was able to successfully login to their account | Pass | - |
| 3 | Test if user is able to select a language | A | User must be logged in to their account | Click | 1)Click on Language Course  2)The Language Course page opens | The Language Course page must open and display it’s contents | The Language Course page is opened and displays it’s contents | Pass | - |
| 4 | Test if app correctly tracks the user progress | A | User must successfully complete a test | User progress | 1)Successfully complete a test | User progress must be correctly updated | User progress is not updated | Fail | Project must be improved to eliminate updation error. |
| 5 | Test if user is able to complete test | A | User must answer the question in test correctly | User’s answer to test | 1)Enter correct answer to question in test. | Message of completion must be displayed | Message of completion is displayed | Pass | - |
| 6 | Test if user is able to retry test on failure | A | User must answer the question in test incorrectly | User’s answer to test | 1)Enter incorrect answer to question in test.  2)Click on retry | Message of failure must be displayed and option of retry must be available | Message of failure is displayed and option to retry is available and works properly | Pass | - |
| 7 | Test if user is able to edit profile details | A | User must be logged in to their account | Username, password, email id, dob | 1)Click on privacy in profile.  2)Enter input (any) username, email id, password and dob on the respective fields  3)click submit | Profile must be updated with new details | Profile is updated with new details | Pass | - |

**B.2 Conclusion**

*(Students must write the conclusion as per the attainment of individual outcome listed above and project definition noted in section B.1 including “Define the System, Motivation, Scope of the System and Applications”)*

*Thus, we successfully designed the test cases for our project, Language Learning App*