1. **DESIGN**
   1. **Introduction**

The purpose of the design phase is to plan a solution of the problem specified by the requirement document. This phase is the first step in moving from the problem domain to the solution domain. In other words, starting with what is needed, design takes us toward how to satisfy the needs. The design of a system is perhaps the most critical factor affection the quality of the software; it has a major impact on the later phase, particularly testing, maintenance. The output of this phase is the design document. This document is similar to a blueprint for the solution and is used later during implementation, testing and maintenance. The design activity is often divided into two separate phases System Design and Detailed Design.

System Design also called top-level design aims to identify the modules that should be in the system, the specifications of these modules, and how they interact with each other to produce the desired results. At the end of the system design all the major data structures, file formats, output formats, and the major modules in the system and their specifications are decided.

During, Detailed Design, the internal logic of each of the modules specified in system design is decided. During this phase, the details of the data of a module is usually specified in a high-level design description language, which is independent of the target language in which the software will eventually be implemented.

In system design the focus is on identifying the modules, where as during detailed design the focus is on designing the logic for each of the modules. In other works, in system design the attention is on what components are needed, while in detailed design how the components can be implemented in software is the issue.

Design is concerned with identifying software components specifying relationships among components. Specifying software structure and providing blue print for the document phase. Modularity is one of the desirable properties of large systems. It implies that the system is divided into several parts. In such a manner , the interaction between parts is minimal clearly specified.

During the system design activities , Developers bridge the gap between the requirements specification , produced during requirements elicitation and analysis , and the system that is delivered to the user.

Design is the place where the quality is fostered in development . Software design is a process through which requirements are translated into a representation of software.

Data Flow Diagram

A graphical tool used to describe and analyze the moment of data through a system manual or automated including the process, stores of data, and delays in the system. Data Flow Diagrams are the central tool and the basis from which other components are developed. The transformation of data from input to output, through processes, may be described logically and independently of the physical components associated with the system. The DFD is also know as a data flow graph or a bubble chart.

DFDs are the model of the proposed system. They clearly should show the requirements on which the new system should be built. Later during design activity this is taken as the basis for drawing the system’s structure charts. The Basic Notation used to create a DFD’s are as follows:

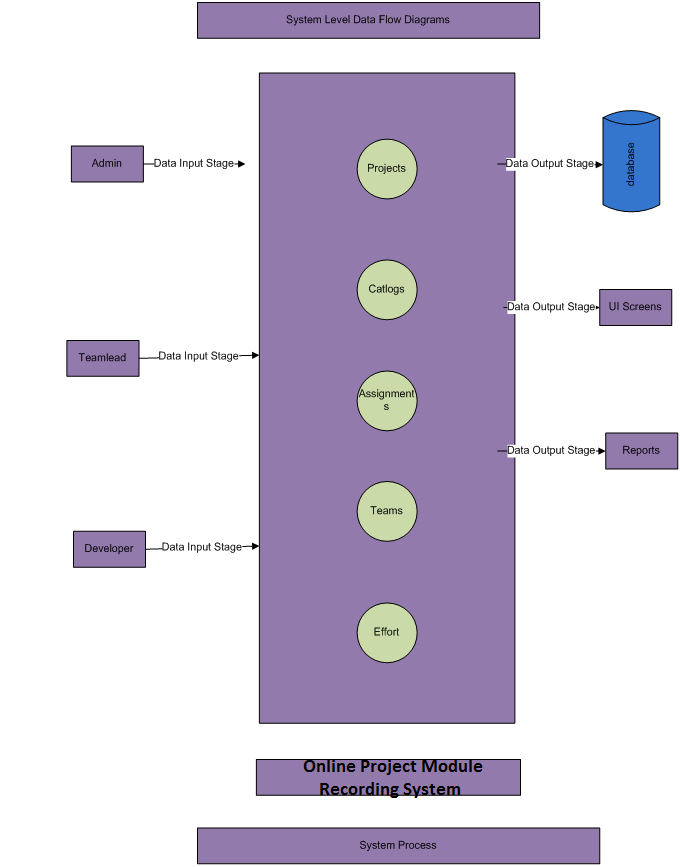
**1. Dataflow:** Data move in a specific direction from an origin to a destination.

**2. Process:** People, procedures, or devices that use or produce (Transform) Data. The physical component is not identified.

**3. Source:** External sources or destination of data, which may be People, programs, organizations or other entities.

**4. Data Store:** Here data are stored or referenced by a process in the System.

**Context Level Data Flow Diagram**



**Level1 Data Flow Diagram for Admin/Manager:**



**Level1 Data Flow Diagram for Project Manager:**



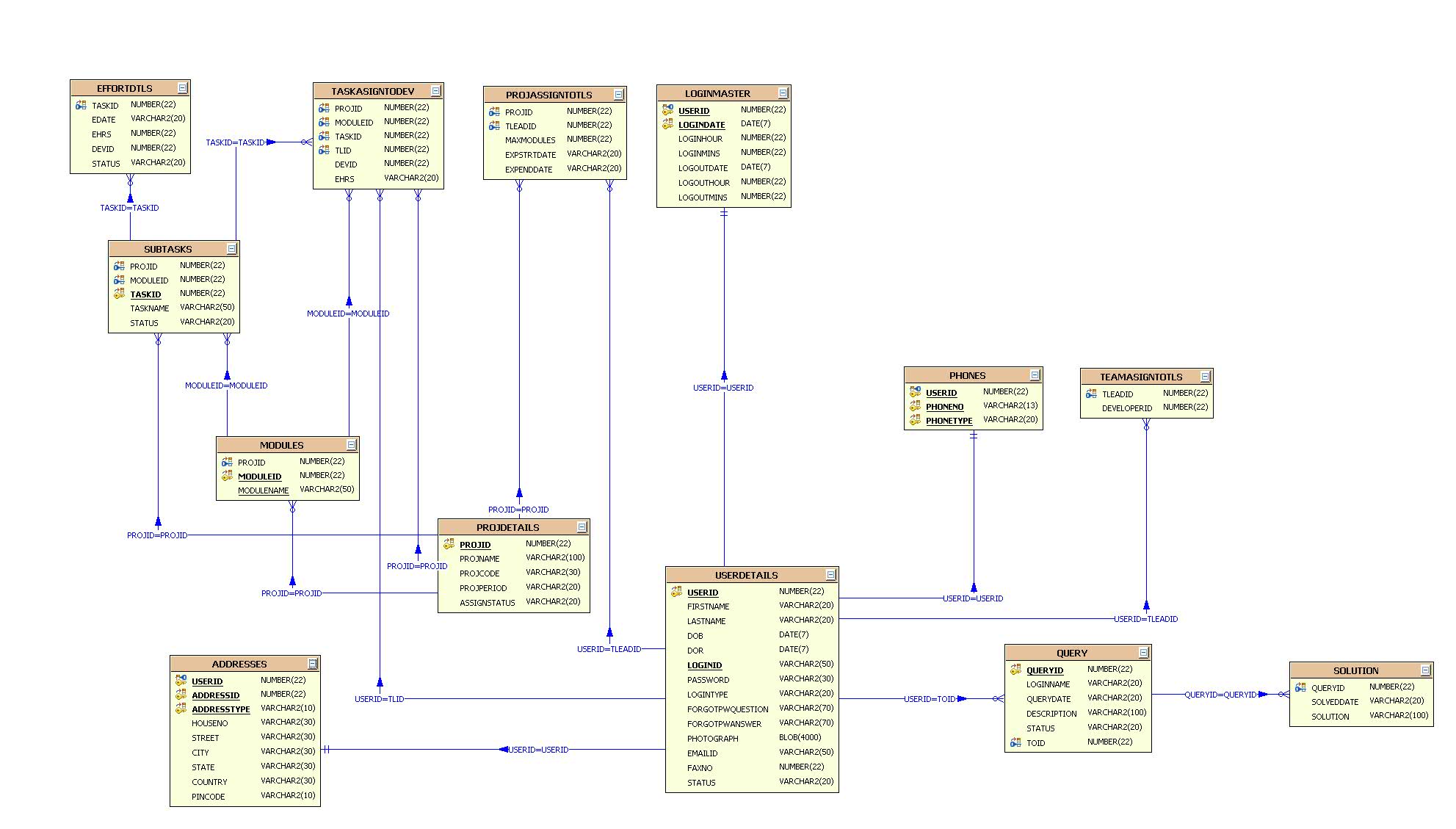
**Level1 Data Flow Diagram for Developer:**



**Authentication Data Flow Diagram:**



**ER Diagram:**

****

**UML DIAGRAMS**

**UNIFIED MODELING LANGUAGE DIAGRAMS**

The unified modeling language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic semantic and pragmatic rules.

A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagram, which is as follows.

**USER MODEL VIEW**

This view represents the system from the users perspective.

The analysis representation describes a usage scenario from the end-users perspective.

**STRUCTURAL MODEL VIEW**

In this model the data and functionality are arrived from inside the system.

This model view models the static structures.

**BEHAVIORAL MODEL VIEW**

It represents the dynamic of behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

**IMPLEMENTATION MODEL VIEW**

In this the structural and behavioral as parts of the system are represented as they are to be built.

**ENVIRONMENTAL MODEL VIEW**

In this the structural and behavioral aspects of the environment in which the system is to be implemented are represented.

UML is specifically constructed through two different domains they are:

UML Analysis modeling, which focuses on the user model and structural model views of the system.

UML design modeling, which focuses on the behavioral modeling, implementation modeling and environmental model views.

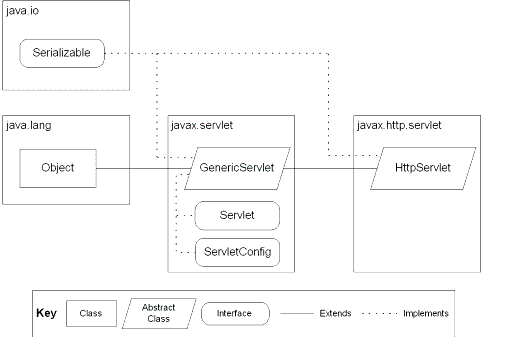
Use case Diagrams represent the functionality of the system from a user’s point of view. Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use cases focus on the behavior of the system from external point of view.

Actors are external entities that interact with the system. Examples of actors include users like administrator, bank customer …etc., or another system like central database.

**Class Diagram**

**CLASS DIAGRAM**

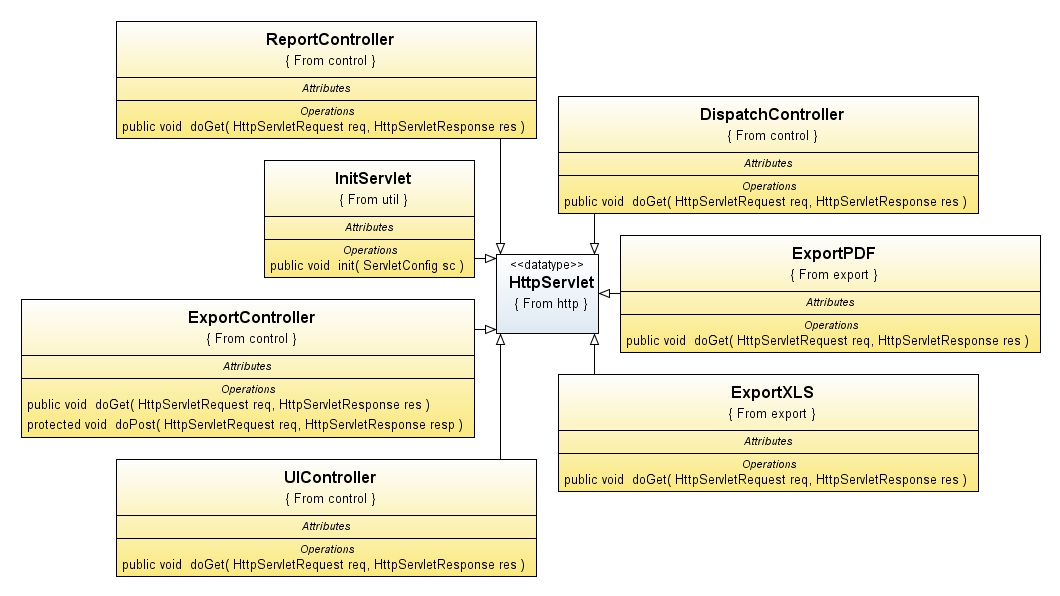
Class diagrams describe the structure of the system in terms of classes and objects. The servlet api class diagram will be as follows.

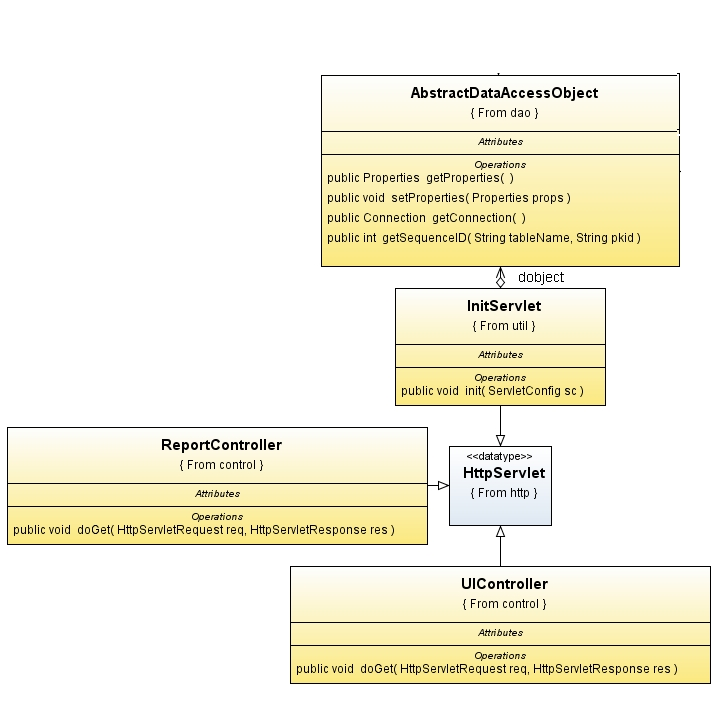


JSP: Implicit Objects

**Class Collaboration Diagrams**

**Class Collaboration Diagram**



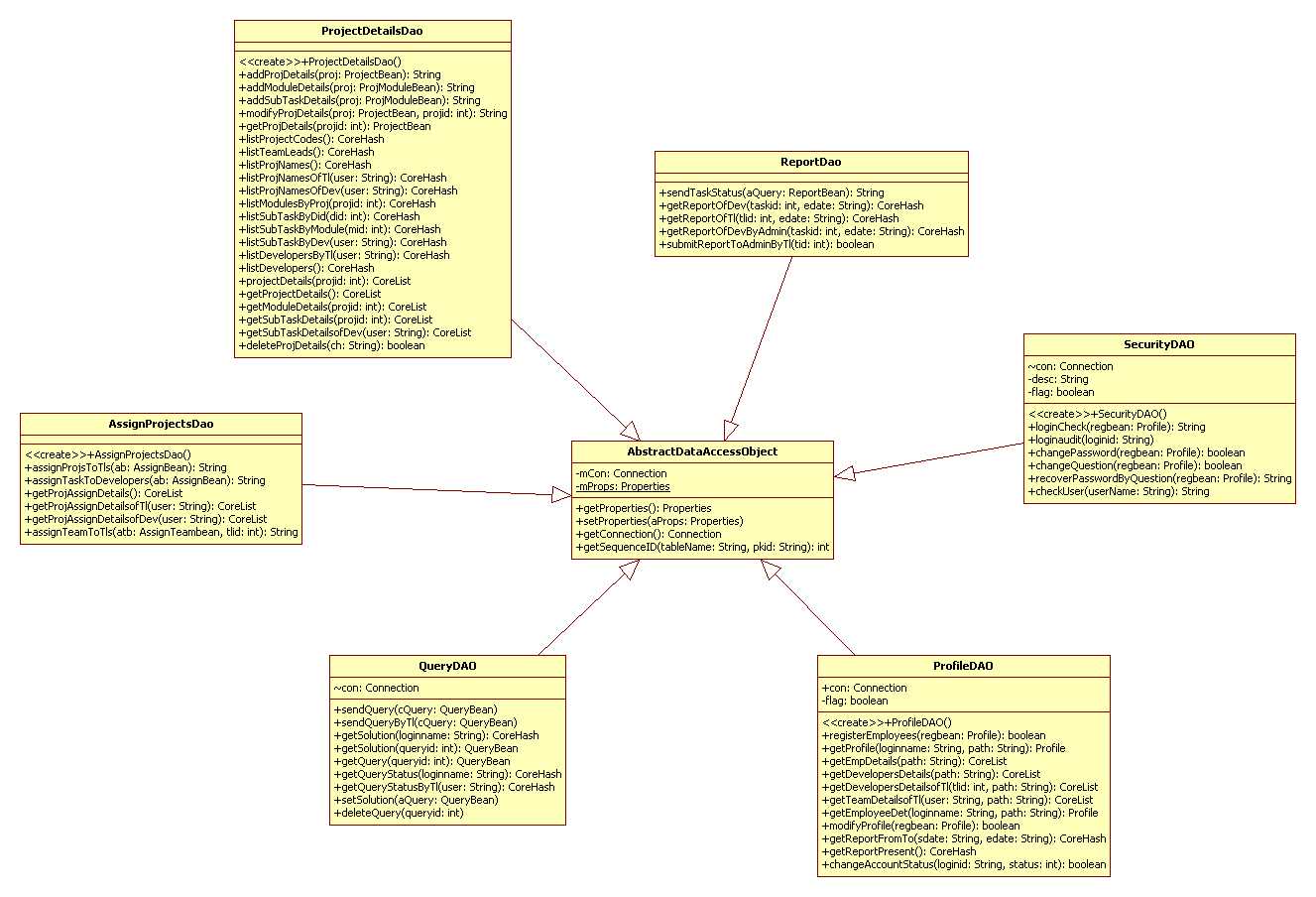












**Use Case Diagrams**

**UML Diagrams**

**Unified Modeling Language**:

The Unified Modeling Language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic semantic and pragmatic rules.

A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagram, which is as follows.

* + User Model View
    1. This view represents the system from the users perspective.
    2. The analysis representation describes a usage scenario from the end-users perspective.
  + Structural model view
    1. In this model the data and functionality are arrived from inside the system.
    2. This model view models the static structures.
* Behavioral Model View

It represents the dynamic of behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.

* Implementation Model View

In this the structural and behavioral as parts of the system are represented as they are to be built.

* Environmental Model View

In this the structural and behavioral aspects of the environment in which the system is to be implemented are represented.

UML is specifically constructed through two different domains they are:

* UML Analysis modeling, this focuses on the user model and structural model views of the system.
* UML design modeling, which focuses on the behavioral modeling, implementation modeling and environmental model views.

Use case Diagrams represent the functionality of the system from a user’s point of view. Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use cases focus on the behavior of the system from external point of view.

Actors are external entities that interact with the system. Examples of actors include users like administrator, bank customer …etc., or another system like central database.

1. **system Use Case Diagram**

System

Online Project Module . Recording System

**Admin**

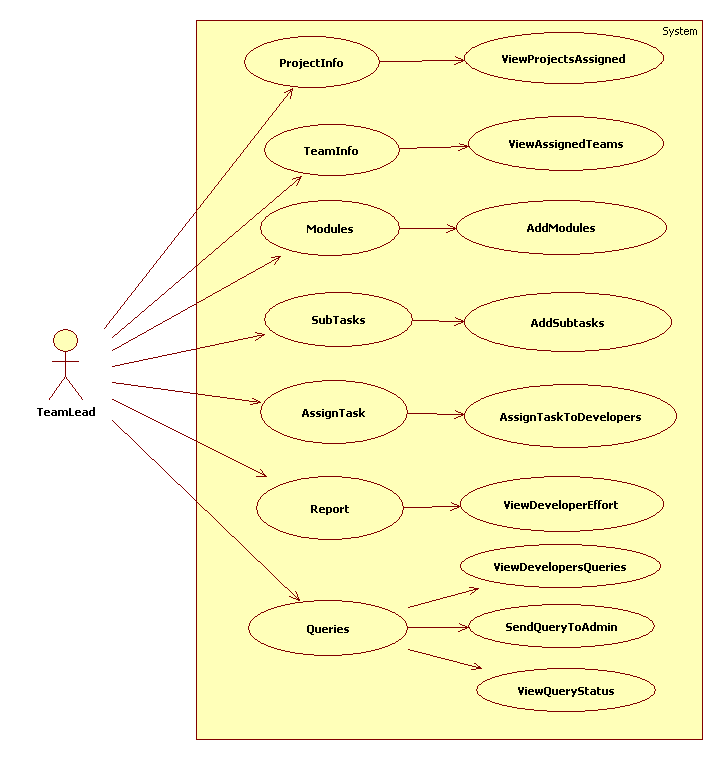
**TeamLead**

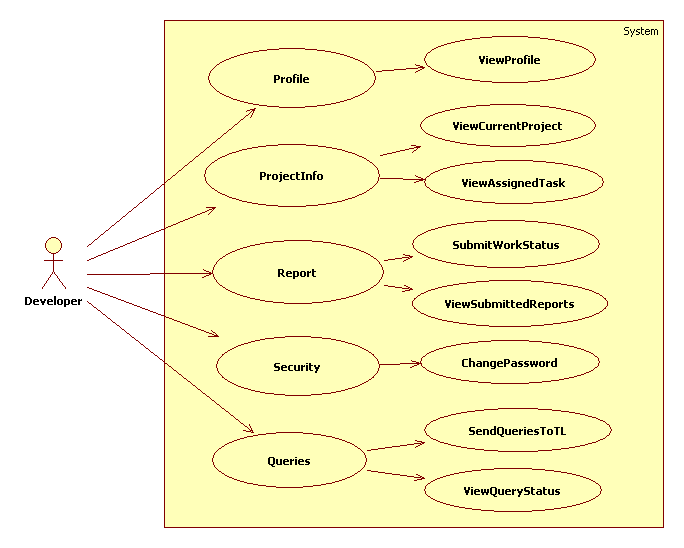
**Developer**

1. **Administrator Use Case Diagram**



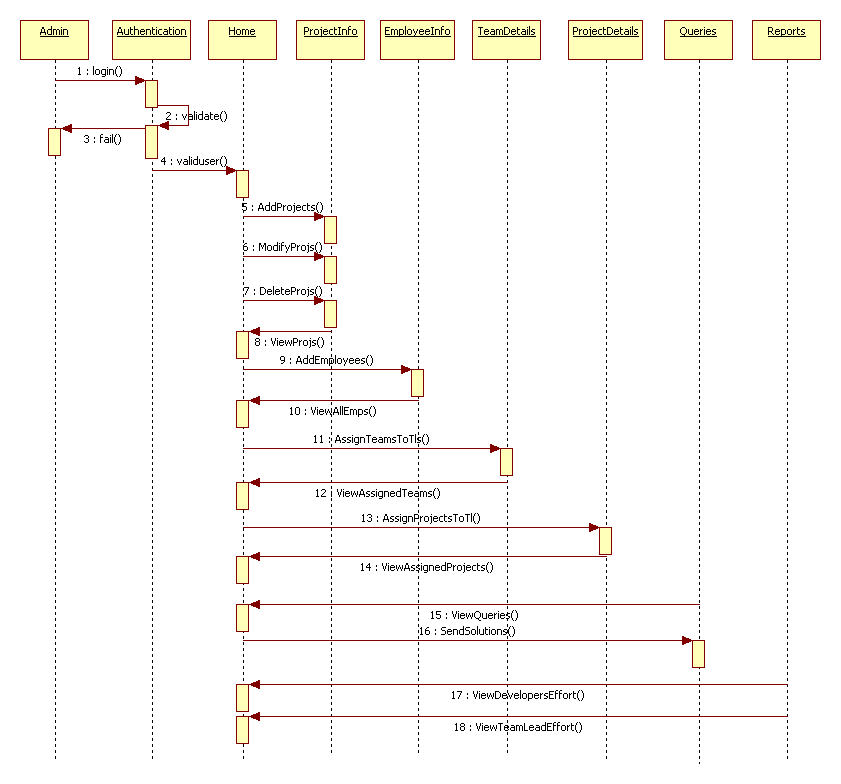
1. **Teamlead Use Case Diagram**



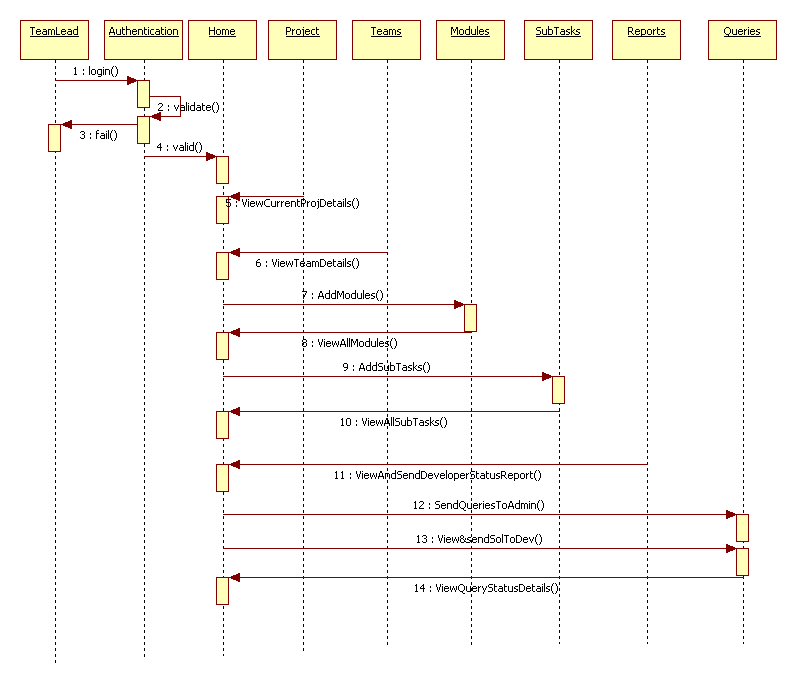
1. **Developer Use Case Diagram**
2. 

**Sequence Diagrams**

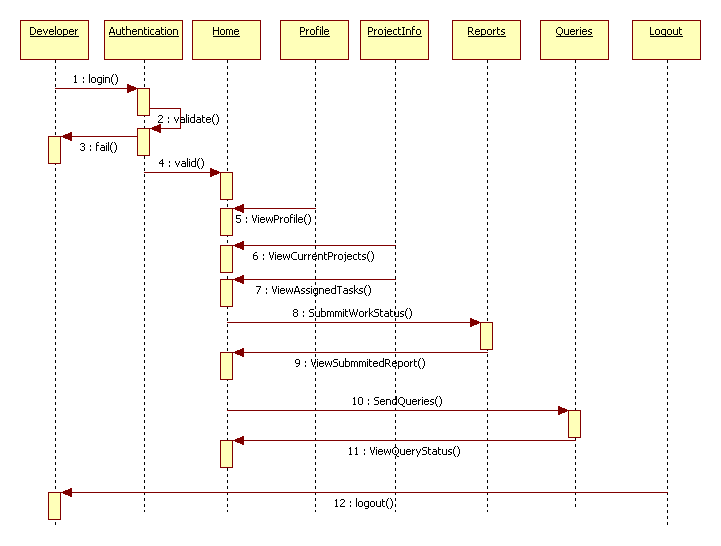
1. **Administrator Sequence Diagram**



1. **Teamlead Sequence Diagram**

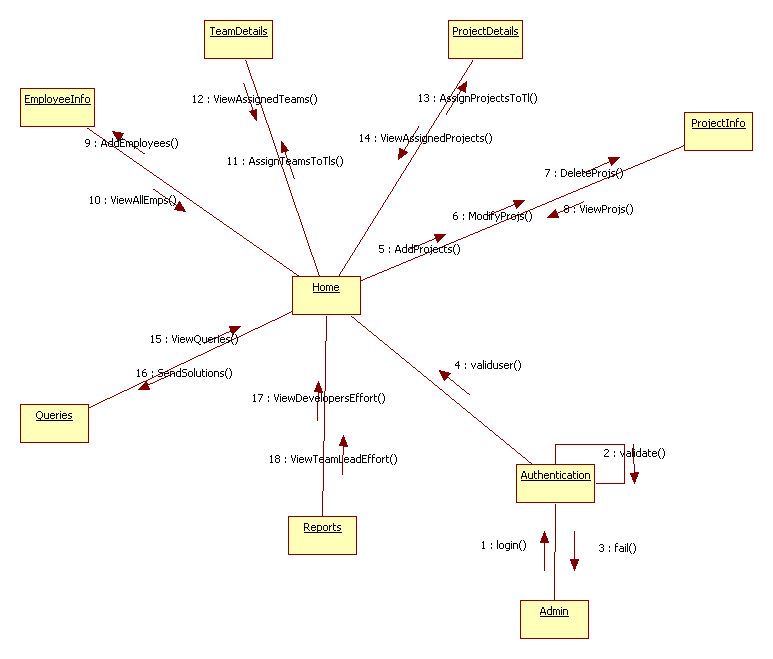


1. **Developer Sequence Diagram**

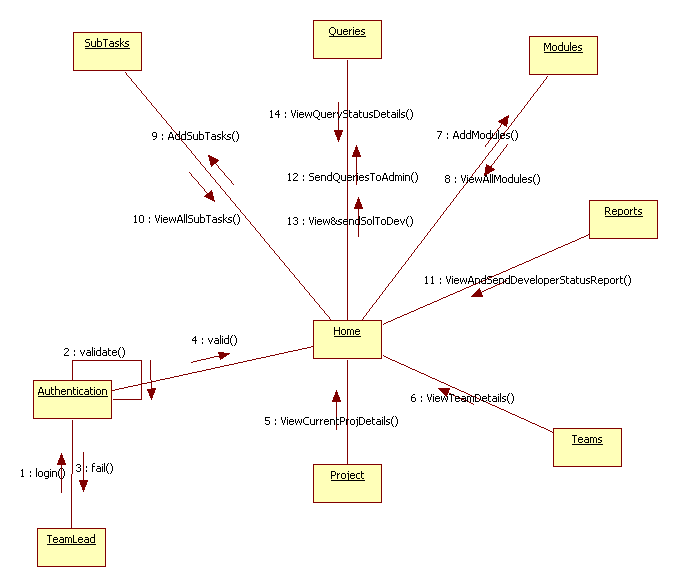


**Collaboration Diagrams**

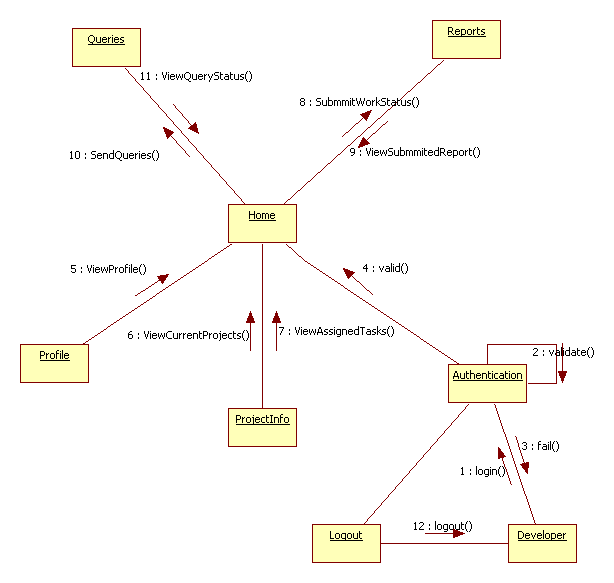
1. **Admin Level CollobrativeDiagram**



1. **Teamlead Level CollobrativeDiagram**



1. **Developer Level CollobrativeDiagram**



**Operation-Level Sequence Diagram**

1. **Login Sequence Diagram**

****

**Login Collaborative Diagram**

****

1. **Present Login User Report Sequence Diagram**

****

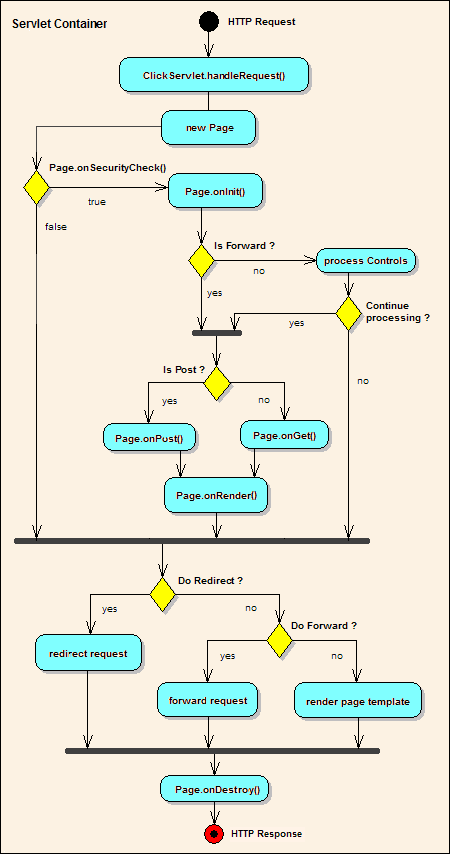
**Present Login User Report Collaborative Diagram**

****

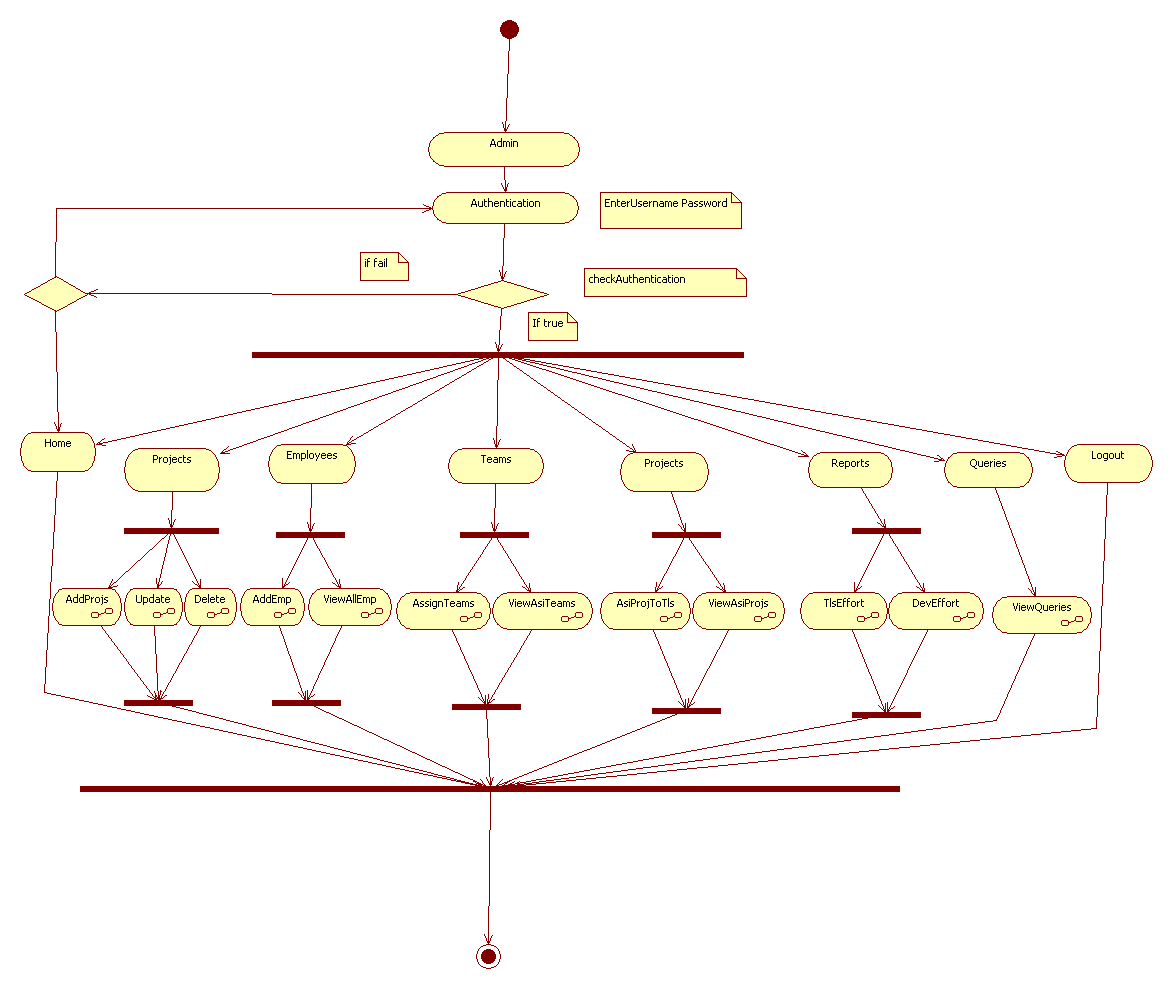
**ACTIVITY DIAGRAMS**

**ACTIVITY DIAGRAMS**

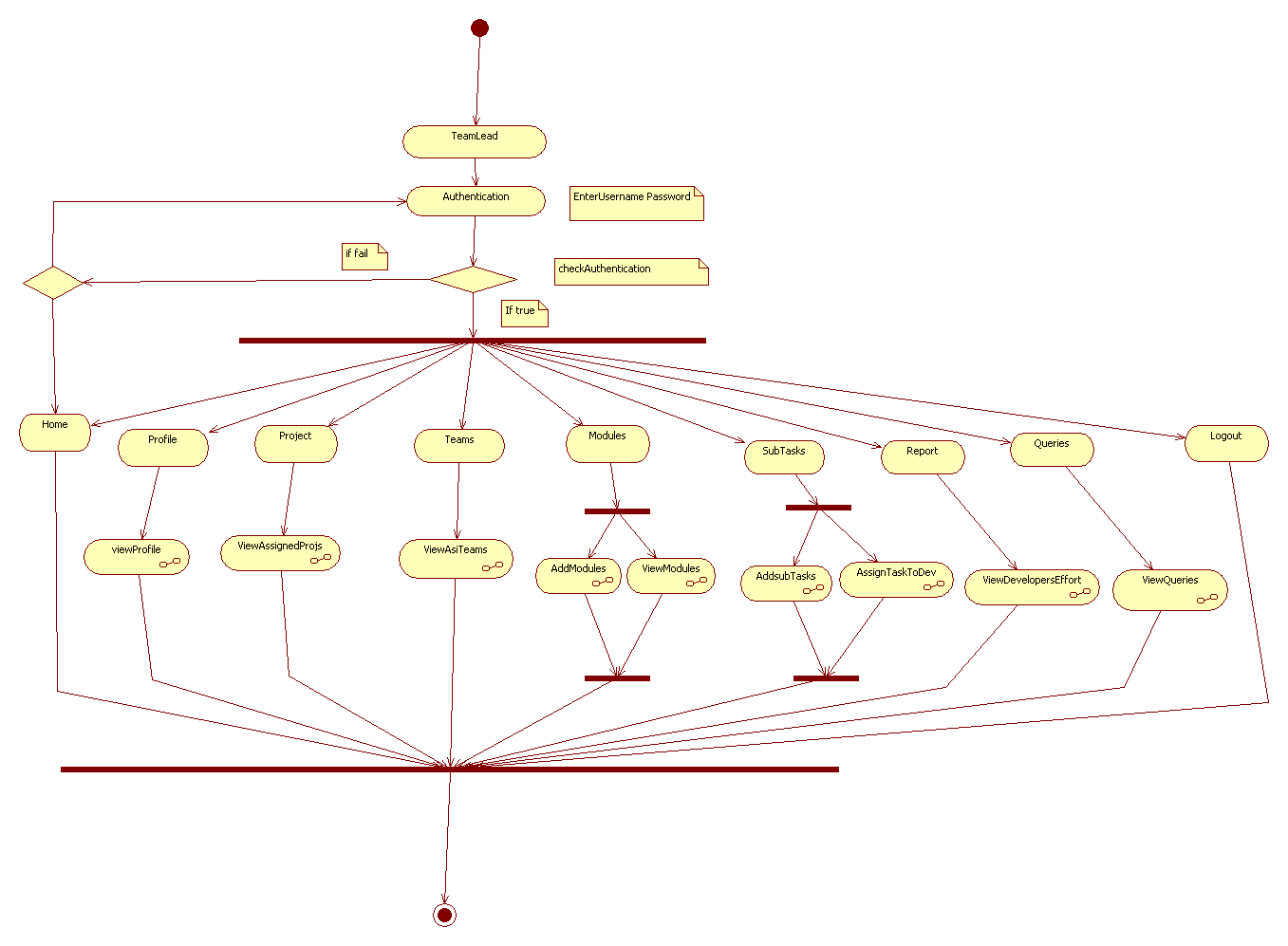
1. **Servlet Container**

****

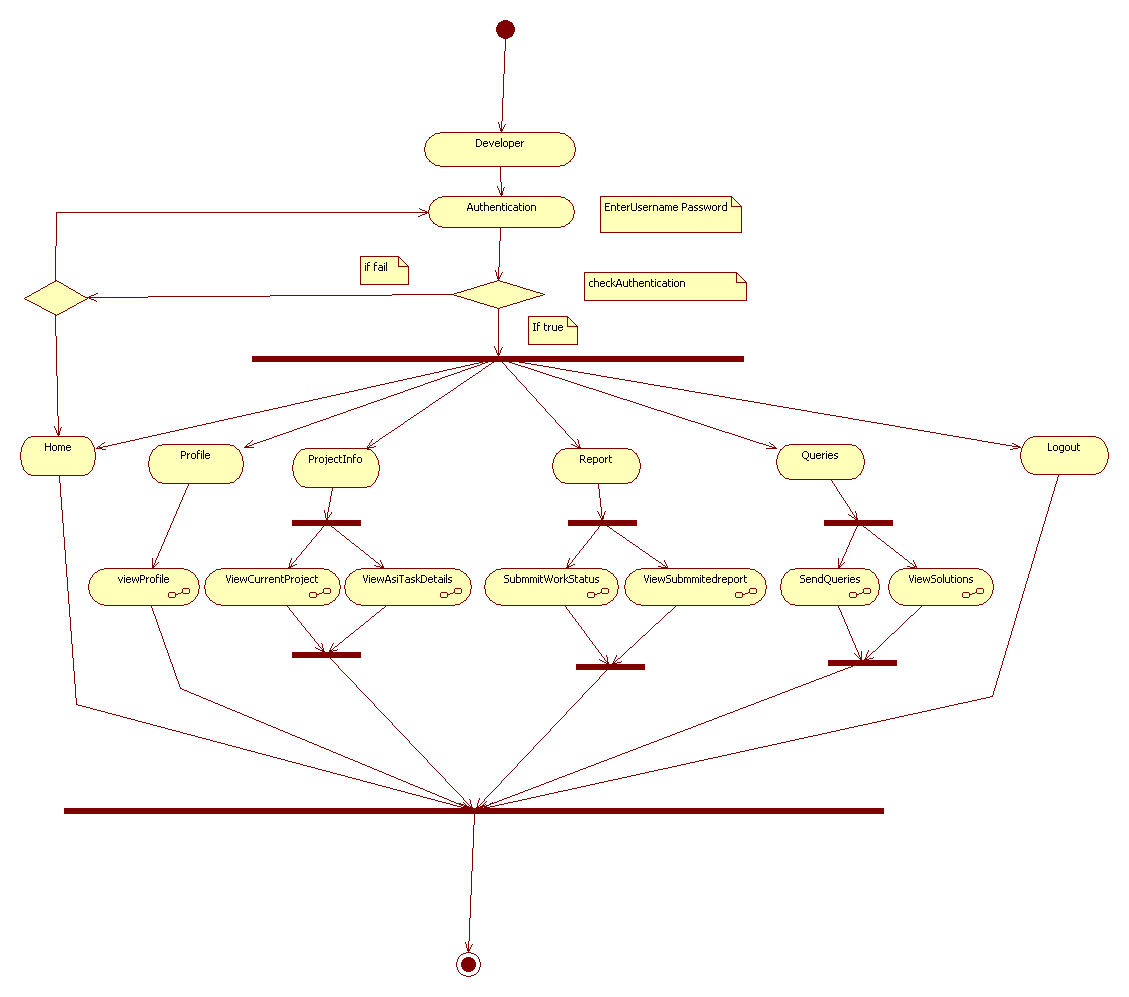
##### Administrator Activity Diagram



##### TeamLead Level Activity Diagram



##### Developer Level Activity Diagram

****

**Component Diagram**

**Component Diagram :**

****

**Deployment Diagram**

**Deployment Diagram:**

****