**UML Design**

**Inside Chapter:**

2.1 UML Design

2.1.1 Introduction of UML

2.1.2 Use case diagram and scenario

2.1.3 Sequence diagram

2.1.4 Activity diagram

2.1.5 Class diagram

2.2 Database Design

2.2.1 Normalization Concept

2.2.2 ERD diagram

**2.1 UML**

**2.1.1 Introduction of UML**

Design is the planning that lays the basis for the making of every object or system. It is the process of originating and developing a plan for a product, structure, system, or component with intention. Is used for either the final (solution) plan (e.g. proposal, drawing, model, description) or the result of implementing that plan in the form of the final product of a design process.

The person designing is called a designer, which is also a term used for people who work professionally in one of the various design areas, usually also specifying which area is being dealt with (such as a fashion designer, concept designer or web designer). Designing often requires a designer to consider the aesthetic, functional, and many other aspects of an object or a process, which usually requires considerable research, thought, modeling, interactive adjustment, and re-design.

**2.1.1.1 UML Concept**

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of software engineering. The standard is managed, and was created by, the Object Management Group.

UML includes a set of graphical notation techniques to create visual models of software-intensive systems. It is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software intensive system under development.

UML combines best techniques from data modeling (entity relationship diagrams), business modeling (work flows), object modeling, and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies. UML aims to be a standard modeling language which can model concurrent and distributed systems.

There are many aspects to consider in the design of a piece of software. The importance of each should reflect the goals the software is trying to achieve. Some of these aspects are:

* Compatibility - The software is able to operate with other products that are designed for interoperability with another product. For example, a piece of software may be backward-compatible with an older version of itself.
* Extensibility - New capabilities can be added to the software without major changes to the underlying architecture.
* Fault-tolerance - The software is resistant to and able to recover from component failure.
* Maintainability - The software can be restored to a specified condition within a specified period of time. For example, antivirus software may include the ability to periodically receive virus definition updates in order to maintain the software's effectiveness.
* Modularity - the resulting software comprises well defined, independent components. That leads to better maintainability. The components could be then implemented and tested in isolation before being integrated to form a desired software system. This allows division of work in a software development project.
* Packaging - Printed material such as the box and manuals should match the style designated for the target market and should enhance usability. All compatibility information should be visible on the outside of the package. All components required for use should be included in the package or specified as a requirement on the outside of the package.
* Reliability - The software is able to perform a required function under stated conditions for a specified period of time.
* Reusability - the modular components designed should capture the essence of the functionality expected out of them and no more or less. This single-minded purpose renders the components reusable wherever there are similar needs in other designs.
* Robustness - The software is able to operate under stress or tolerate unpredictable or invalid input. For example, it can be designed with a resilience to low memory conditions.
* Usability - The software user interface must be usable for its target user/audience.

**2.1.1.2 Goals of UML**

The primary goals in the design of the UML were:

* Provide users with a ready-to-use, expressive visual modeling language so they can develop and exchange meaningful models.
* Provide extensibility and specialization mechanisms to extend the core concepts.
* Be independent of particular programming languages and development processes.
* Provide a formal basis for understanding the modeling language.
* Encourage the growth of tools market.
* Support higher-level development concepts such as collaborations, frameworks, patterns and components.
* Integrate best practices.

**2.1.2 Use case**

* Diagram that shows the relationships among actors and use cases within a system and describes a sequence of actions that provide a measurable value to an actor.

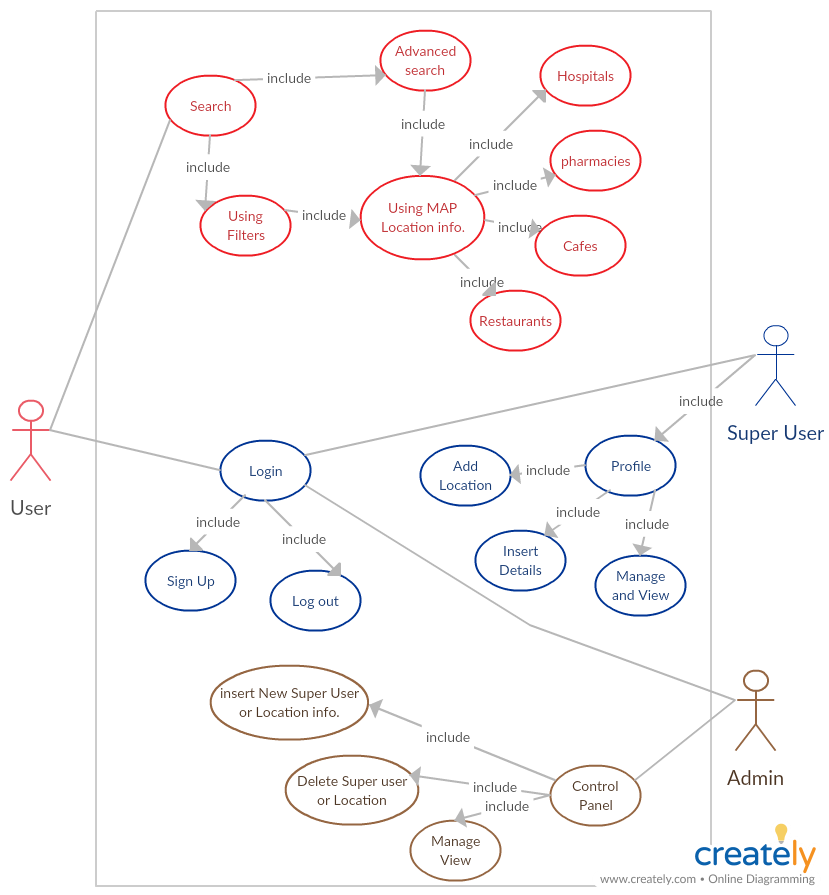


Figure 2.1 – System Use case UML Diagram

**Scenarios**

|  |  |
| --- | --- |
| **Use Case:** | Login |
| **Description:** | Enter email and password to create profile and access the entire system and dashboard |
| **Actor:** | User and Super user |
| **Precondition:** | Registration |
| **Postcondition:** | Location Creation and Profile Creation |

|  |  |
| --- | --- |
| **Use Case:** | Search Nearest Item Location |
| **Description:** | Item Search to get the nearest location that contain this item |
| **Actor:** | User |
| **Precondition:** | MAP Location |
| **Post Condition:** | Get the nearest Location and get profile page of this location |

|  |  |
| --- | --- |
| **Use Case:** | Profile Page |
| **Description:** | Page contains the location and personal information about the super user |
| **Actor:** | Super user |
| **Precondition:** | Registration or Login |
| **Post Condition:** | Edit Information  Update logo image  Update Location  Add or manage Items |

|  |  |
| --- | --- |
| **Use Case:** | Rating Process |
| **Description:** | Rate from 0 to 5 star to super user profile page based on reliability |
| **Actor:** | User |
| **Precondition:** | Registration or login and enter super user Profile |
| **Post Condition:** | Update super user view and rating |

|  |  |
| --- | --- |
| **Use Case:** | MAP Location |
| **Description:** | Get Accurate Location of user on a MAP using Google MAP |
| **Actor:** | Super User or User |
| **Precondition:** | Get IP from IP finder API |
| **Post Condition:** | Create marker on a MAP |

**2.1.3 Sequence**

* Definition
* An interaction diagram that shows the objects participating in a particular interaction and the messages they exchange arranged in a time sequence.
* Are used primarily to design, document and validate the architecture, interfaces and logic of the system by describing the sequence of actions that need to be performed to complete a task or scenario.
* Are useful design tools because they provide a dynamic view of the system behavior which can be difficult to extract from static diagrams or specifications.
* A sequence diagram shows, as parallel vertical lines (lifelines), different processes or objects that live simultaneously, and, as horizontal arrows, the messages exchanged between them, in the order in which they occur. This allows the specification of simple runtime scenarios in a graphical manner.
* Advantages of sequence diagram:
* Are useful design tools because they provide a dynamic view of the system behavior which can be difficult to extract from static diagrams or specifications.
* Help you discover architectural, interface and logic problems early.
* Sequence diagram editor makes it so easy to edit your sequence diagrams that you could even make the corrections in real time during the meeting and instantly see the result of the changes as you make them.
* Documentation. Sequence diagrams can be used to document the dynamic view of the system design at various levels of abstraction, which is often difficult to extract from static diagrams or even the complete source code. The diagrams can abstract much of the implementation detail and provide a high-level view of system behavior.

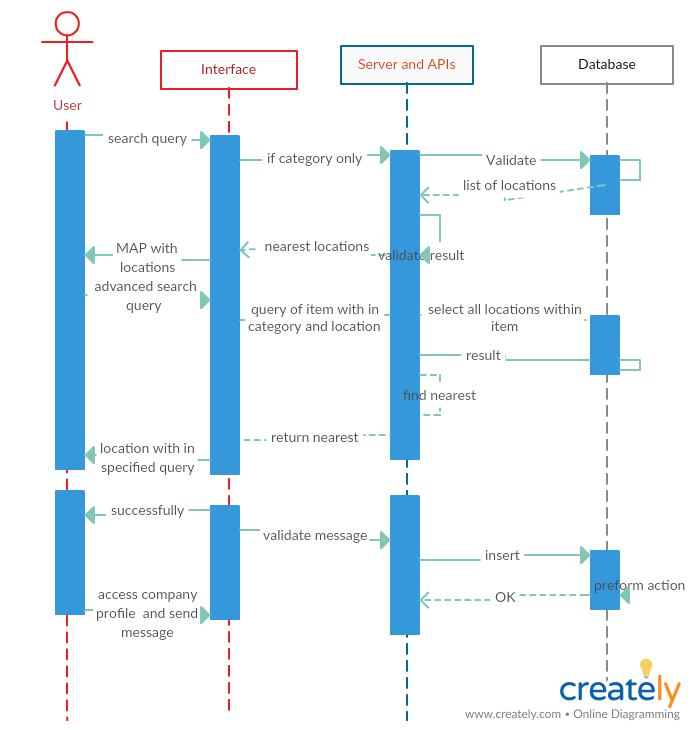
****

Figure 2.2 –User Sequence Diagram

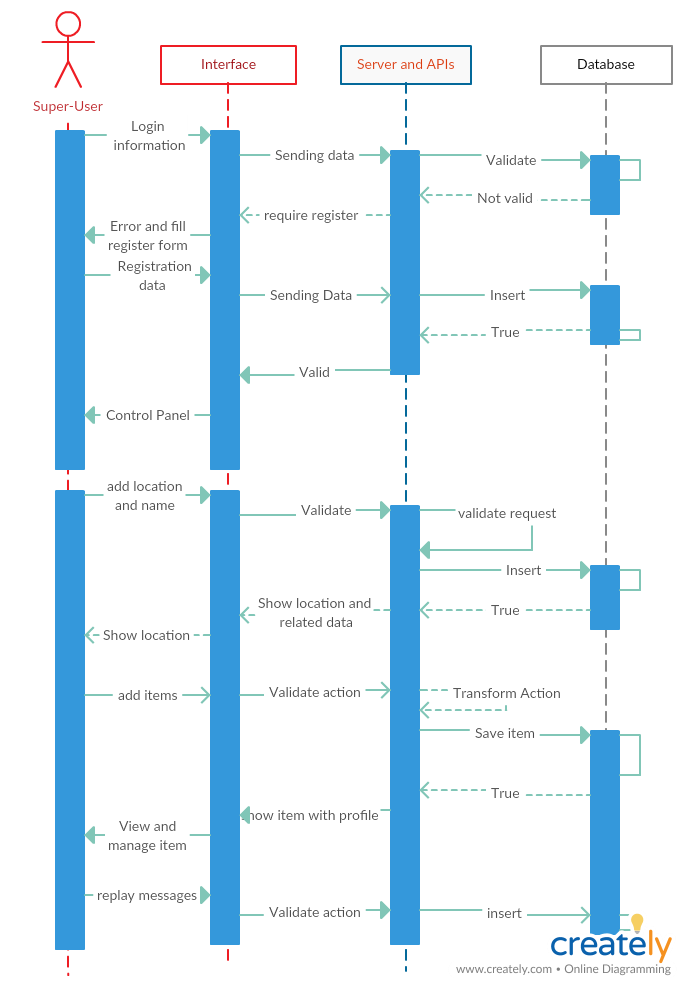


Figure 2.3 – Super user Sequence Diagram

**2.1.4 Activity**

* Definition:
* Activity diagrams are graphical representations of [workflows](http://en.wikipedia.org/wiki/Workflow) of stepwise activities and actions

with support for choice, iteration and concurrency.

* Illustrates the dynamic nature of a system by modeling the flow of control from activity to activity.
* Diagram describe the workflow behavior of a system and can show activities that are conditional or parallel.
* Why we use activity diagrams:
* To model the workflow behind the system being designed.
* Analyzing a use case by describing what actions need to take place.
* Describing a Complicated Sequential Algorithm.
* Modeling applications with parallel processes.

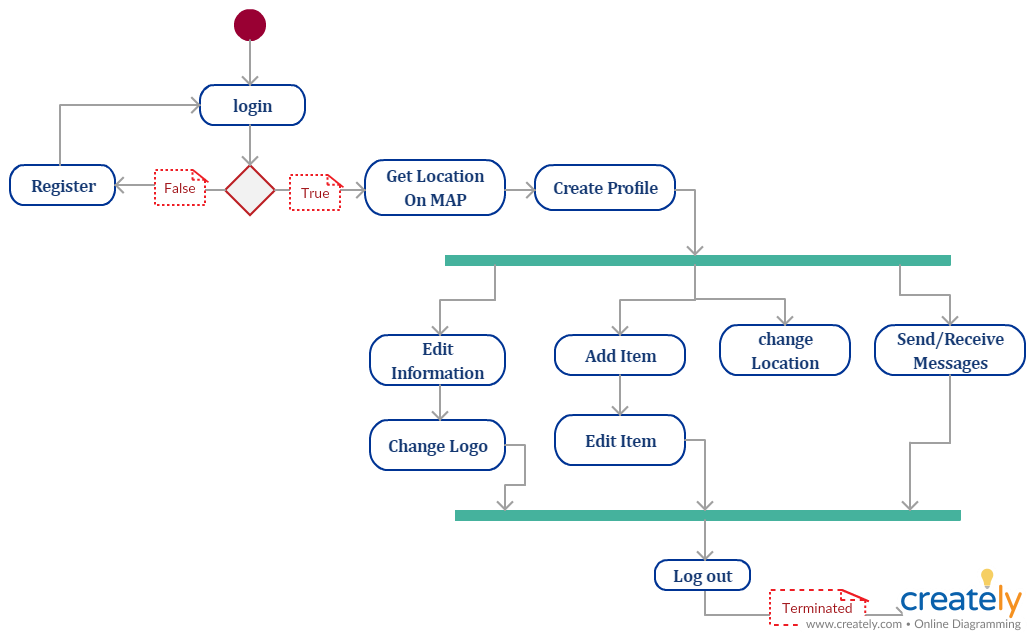


Figure 2.4 – Super user Activity Diagram

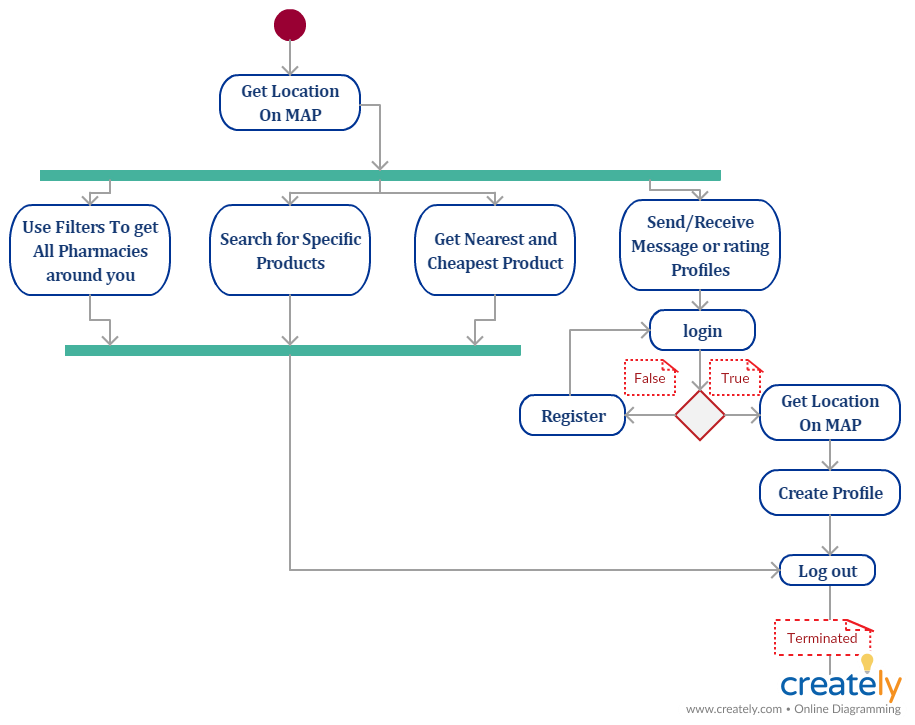


Figure 2.5 –User Activity Diagram

**2.1.5 Class Diagram**

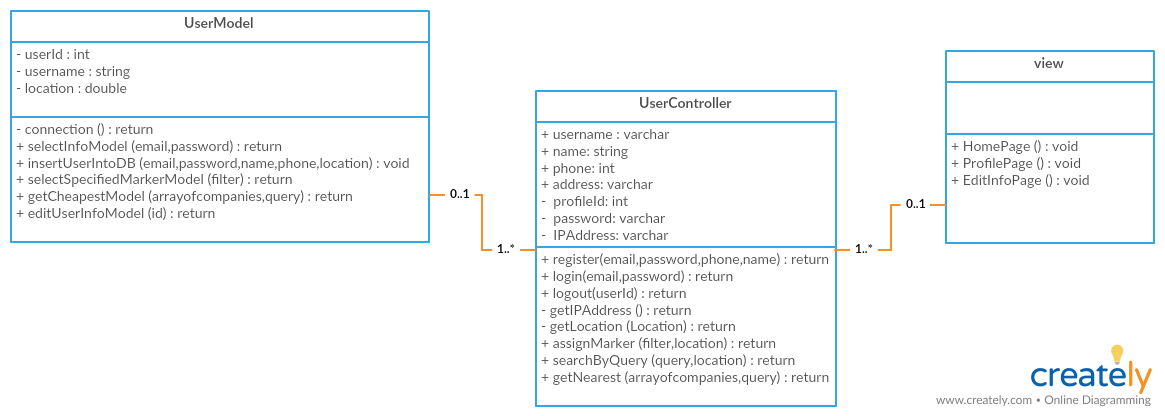


Figure 2.6 –User Class Diagram

**2.2. Database Design:**

* Database design is the process of producing a detailed data model of a database. This logical data model contains all the needed logical and physical design choices and physical storage parameters needed to generate a design in a Data Definition Language, which can then be used to create a database. A fully attributed data model contains detailed attributes for each entity
* The term database design can be used to describe many different parts of the design of an overall database system. Principally, and most correctly, it can be thought of as the logical design of the base data structures used to store the data. In the relational model these are the tables and views. In an object database the entities and relationships map directly to object classes and named relationships. However, the term database design could also be used to apply to the overall process of designing, not just the base data structures, but also the forms and queries used as part of the overall database application within the database management system (DBMS).
* **A database designer** is a person responsible for the design, implementation, maintenance and repair of an organization's database. They are also known by the titles Database Coordinator or Database Programmer, and is closely related to the Database Analyst, Database Modeler, Programmer Analyst, and Systems Manager. The role includes the development and design of database strategies, monitoring and improving database performance and capacity, and planning for future expansion requirements.

**2.2.1 Normalization**

Normalization refers to the process of structuring data in order to minimize duplication and inconsistencies. There are several types of normalization:

* First Normal Form
* The rules for the first normal form are as follows:
* Eliminate repeating information.
* Create separate tables for related data.
* Breaking the flat table into two tables.
* Second Normal Form
* The rule for the second normal form is as follows:
  + No non-key attributes depend on portion of the primary key
* Third Normal Form
* The rule for the third normal form is as follows:
* No attributes depend on other non-key attributes.

Relationships:

A relationship works by matching data in key columns — usually columns with the same name in both tables. In most cases, the relationship matches the primary key from one table, which provides a unique identifier for each row, with an entry in the foreign key in the other table.

There are three types of relationships between tables. The type of relationship that is created depends on how the related columns are defined.

# One-to-Many Relationships

# A one-to-many relationship is the most common type of relationship. In this type of relationship, a row in table A can have many matching rows in table B, but a row in table B can have only one matching row in table A.

# Many-to-Many Relationships

* In a many-to-many relationship, a row in table A can have many matching rows in table B, and vice versa. You create such a relationship by defining a third table, called a junction table, whose primary key consists of the foreign keys from both table A and table B.

# One-to-One Relationships

* In a one-to-one relationship, a row in table A can have no more than one matching row in table B, and vice versa. A one-to-one relationship is created if both of the related columns are primary keys or have unique constraints.

**2.2.2 ERD (Entity Relationship Diagram)**

