Daily Checklist & Daily Routine iOS Application

A PROJECT REPORT

Submitted by

Rajat Joshi

Dixit Baravaliya

Arthkumar Patel

Bhgaya Patel

In fulfillment for the award of the degree

of

BACHELOR OF ENGINEERING

in

Computer Engineering

LDRP Institute of Technology and Research, Gandhinagar

Kadi Sarva Vishwavidyalaya

April/May, 2020

LDRP INSTITUTE OF TECHNOLOGY AND RESEARCH

GANDHINAGAR

CE-IT Department

CERTIFICATE

This is to certify that the Project Work entitled **"Daily Checklist & Daily Routine iOS Application"** has been carried out by **Joshi Rajat (1620BECE30150)** under my guidance in fulfilment of the degree of Bachelor of Engineering in Computer Engineering (8th Semester) of Kadi Sarva Vishwavidyalaya during the academic year 2019- 2020.

Prof. Sandip Modha	Dr. Shivangi Surati
Internal Guide	**Head of the Department**
LDRP ITR	**LDRP ITR**

Kadi Sarva Vishwavidyalaya

April/May, 2020

LDRP INSTITUTE OF TECHNOLOGY AND RESEARCH

GANDHINAGAR

CE-IT Department

CERTIFICATE

This is to certify that the Project Work entitled **"Daily Checklist & Daily Routine iOS Application"** has been carried out by **Dixit Baravaliya (1620BECE30013)** under my guidance in fulfilment of the degree of Bachelor of Engineering in Computer Engineering (8th Semester) of Kadi Sarva Vishwavidyalaya during the academic year 2019- 2020.

Prof. Sandip Modha	Dr. Shivangi Surati
Internal Guide	**Head of the Department**
LDRP ITR	**LDRP ITR**

Kadi Sarva Vishwavidyalaya

April/May, 2020

LDRP INSTITUTE OF TECHNOLOGY AND RESEARCH

GANDHINAGAR

CE-IT Department

CERTIFICATE

This is to certify that the Project Work entitled **"Daily Checklist & Daily Routine iOS Application"** has been carried out by **Arthkumar Patel (1620BECE30007)** under my guidance in fulfilment of the degree of Bachelor of Engineering in Computer Engineering (8th Semester) of Kadi Sarva Vishwavidyalaya during the academic year 2019- 2020.

Prof. Sandip Modha	Dr. Shivangi Surati
Internal Guide	**Head of the Department**
LDRP ITR	**LDRP ITR**

Kadi Sarva Vishwavidyalaya

April/May, 2020

LDRP INSTITUTE OF TECHNOLOGY AND RESEARCH

GANDHINAGAR

CE-IT Department

CERTIFICATE

This is to certify that the Project Work entitled **"Daily Checklist & Daily Routine iOS Application"** has been carried out by **Bhagya Patel (1620BECE30009)** under my guidance in fulfilment of the degree of Bachelor of Engineering in Computer Engineering (8th Semester) of Kadi Sarva Vishwavidyalaya during the academic year 2019- 2020.

Prof. Sandip Modha	Dr. Shivangi Surati
Internal Guide	**Head of the Department**
LDRP ITR	**LDRP ITR**

TABLE OF CONTENTS:

4.4 Feasibility Study

4.4.1 Does the system contribute to the overall objectives of the organization?

4.4.2 Can the system be implemented using the current technology and within the given cost and schedule constraints?

4.4.3 Can the system be integrated with other system which is already in place?

4.5 Requirements Validation(is concerned with showing that the requirements actually define the system which the customer wants)

4.6 Activity/Process In New System(Use event table)

4.7 Features Of New System

4.8 Class Diagram

4.9 System Activity(Use case and/or scenario diagram)

4.10 Object Interaction

4.11 Sequence and Collaboration Diagram

5 System Design

5.1 System Application Design

5.1.1 Method Pseudo code

5.2 Database Design/Data Structure Design

5.2.1 Table and Relationship

5.2.2 Logical Description Of Data

5.3 Input/Output and Interface Design

5.3.1 State Transition/UML Diagram

5.3.2 Samples Of Forms, Reports and Interface

6 System Testing

Test cases

7 Conclusion

8 Bibliography

Acknowledgement

We are pleased to present “**Daily Checklist and Daily Routine App**” project and take this opportunity to express our profound gratitude to all those people who helped us in completion of this project.

We thank our college for providing us with excellent facilities that helped us to complete and present this project. We would also like to thank the staff members and lab assistants for permitting us to use computers in the lab as and when required.

We express our deepest gratitude towards our project guide for his/her valuable and timely advice during the various phases in our project. We would also like to thank him/her for providing us with all proper facilities and support as the project co-coordinator. We would like to thank him/her for support, patience and faith in our capabilities and for giving us flexibility in terms of working and reporting schedules.

We would like to thank all our friends for their smiles and friendship making the college life enjoyable and memorable and family members who always stood beside us and provided the utmost important moral support. Finally, we would like to thank everyone who has helped us directly or indirectly in our project.

PROJECT OVERVIEW

Introduction

Nowadays everyone has busy schedule, and many don’t have time to organize their everyday life, people face difficulties to manage their daily routines, to schedule time and focus on big tasks. To overcome these issues, we have come up with this daily routine application which will helps us to schedule our daily routine and maintain daily checklist. In this to-do list app, we can update our daily routine as well as weekly tasks. We can also delete and add the task from morning to night and set reminder, we can also see the previous task performed in previous months, weeks and days. We get reminders for a particular task through notification. This is how, we can schedule our tasks as per our timing and will help us to remind and complete every task in an easy and efficient way.

Scope and Objective

Everyone has busy schedule, and many don’t have time to organize their everyday life, people face difficulties to manage their daily routines, to schedule time and focus on big tasks. To overcome these issues, we have come up with this daily routine application which will helps us to schedule our daily routine and maintain daily checklist. Here we can update our daily routine as well as weekly tasks. We can also delete and add the task from morning to night and set reminder, we can also see the previous task performed in previous months, weeks and days. They will get notification or Reminder of the Task.

Modules and their Description

The system comprises of 1 major modules and other sub modules as follows:

1. **User**

- **Home**: Home has Tabs with Days of Week Listing such from Monday to Sunday which symbolizes current week.
- **Day of the Week**: Every Day of the Week has its Task Mentioned which can be set to Completed.
- **Manage Task**: User can add, update or delete a Task.
- **Reminders:** While Adding Tasks user can set a Reminder of the corresponding task.
- **Task History:** user can view his Daily Routine for a Particular day which doesn't fall on the current week.
- **Notification:** Notification or Reminder of the Task.

Existing System & Proposed System

❖ Problem with current scenario

- People nowadays have busy schedules many times they missed out doing small but important things.
- Many people feel like they are stressed, anxious, overwhelmed and falling short of their goals and true potential.

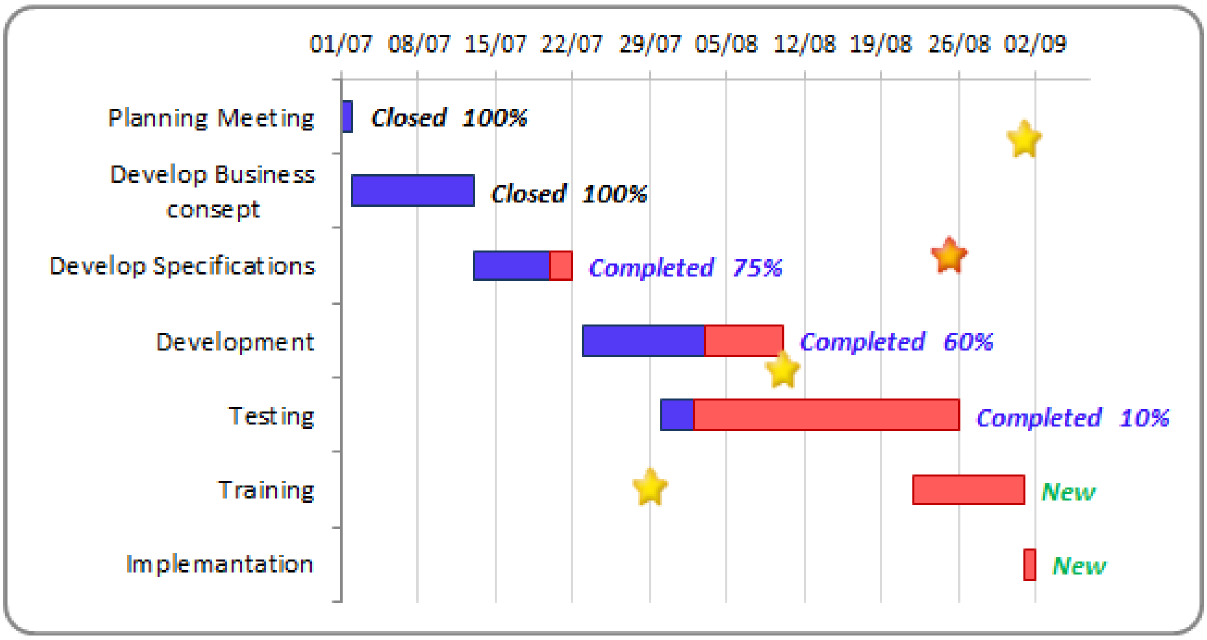
Drawbacks of the existing system

- Maintenance of the system is very difficult.
- There is a possibility for getting inaccurate results.
- User friendliness is very less.
- It consumes more time for processing the task.

PROPOSED SYSTEM

- Considering the anomalies in the existing system computerization of the whole activity is being suggested after initial analysis.
- Proposed system is accessed by one entity namely, User.
- User need to login with their valid login credentials first in order to access the android application.
- After successful login, User can access all the modules and perform/manage each task accurately.
- User can perform task such as daily routine application which will helps us to schedule our daily routine and maintain daily checklist.
- In this to-do list app, we can update our daily routine as well as weekly tasks.
- We can also delete and add the task from morning to night and set reminder, we can also see the previous task performed in previous months, weeks and days.
- We get reminders for a particular task through notification. This is how, we can schedule our tasks as per our timing and will help us to remind and complete every task in an easy and efficient way.

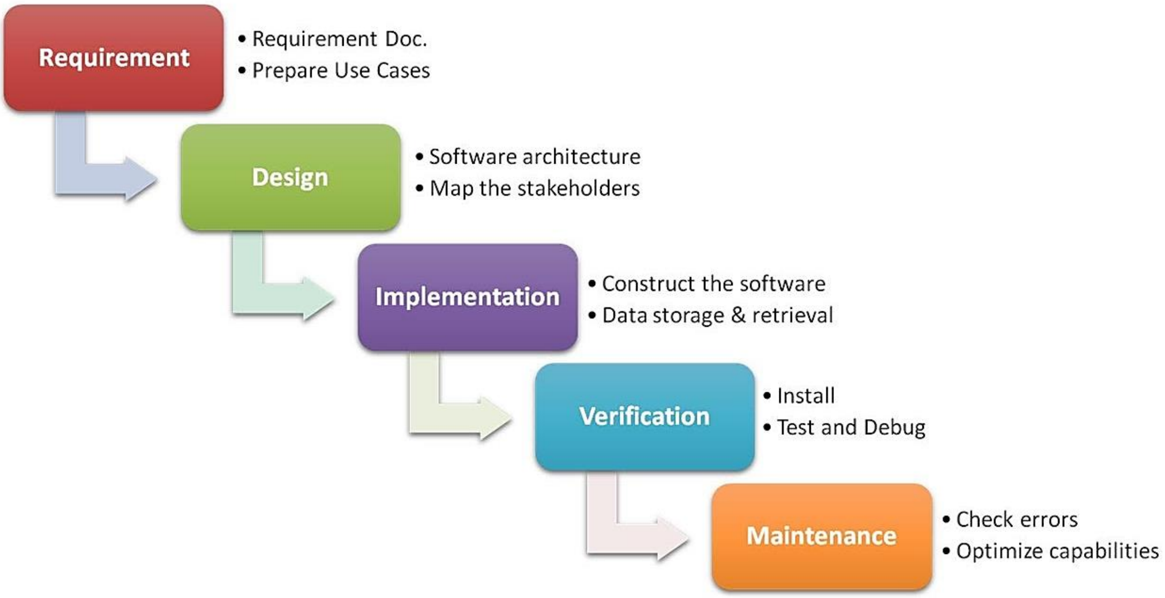
Gantt Chart



01/07
08/07
15/07
22/07
29/07
05/08
12/08
19/08
26/08
02/09
Planning Meeting
Closed 100%
Develop Business consept
Closed 100%
Develop Specifications
Completed 75%
Development
Completed 60%
Testing
Completed 10%
Training
New
Implamantation
New

Project Lifecycle Details

Waterfall Model

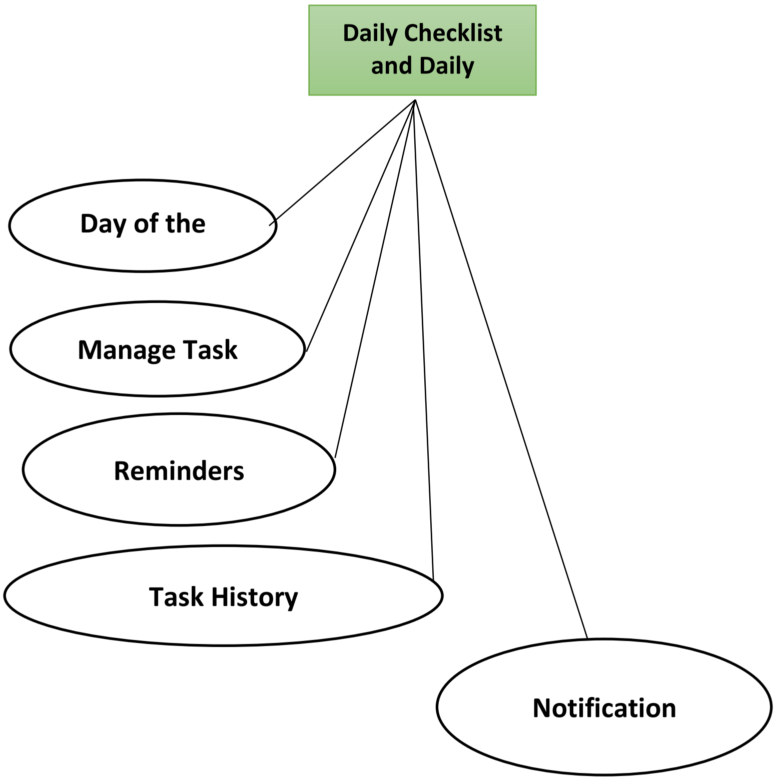


Description

The waterfall Model is a linear sequential flow. In which progress is seen as flowing steadily downwards (like a waterfall) through the phases of software implementation. This means that any phase in the development process begins only if the previous phase is complete. The waterfall approach does not define the process to go back to the previous phase to handle changes in requirement. The waterfall approach is the earliest approach that was used for software development.

PROJECT DESIGN

E-R Diagram



Use Case Diagram

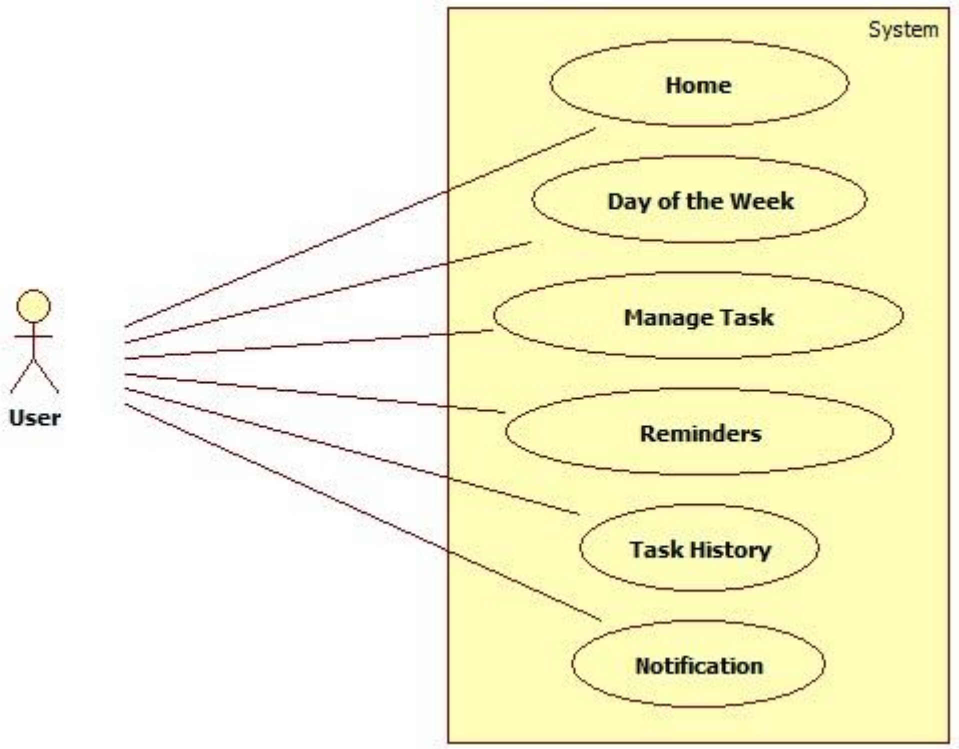


Fig. Use Case Diagram of User

Sequence Diagram

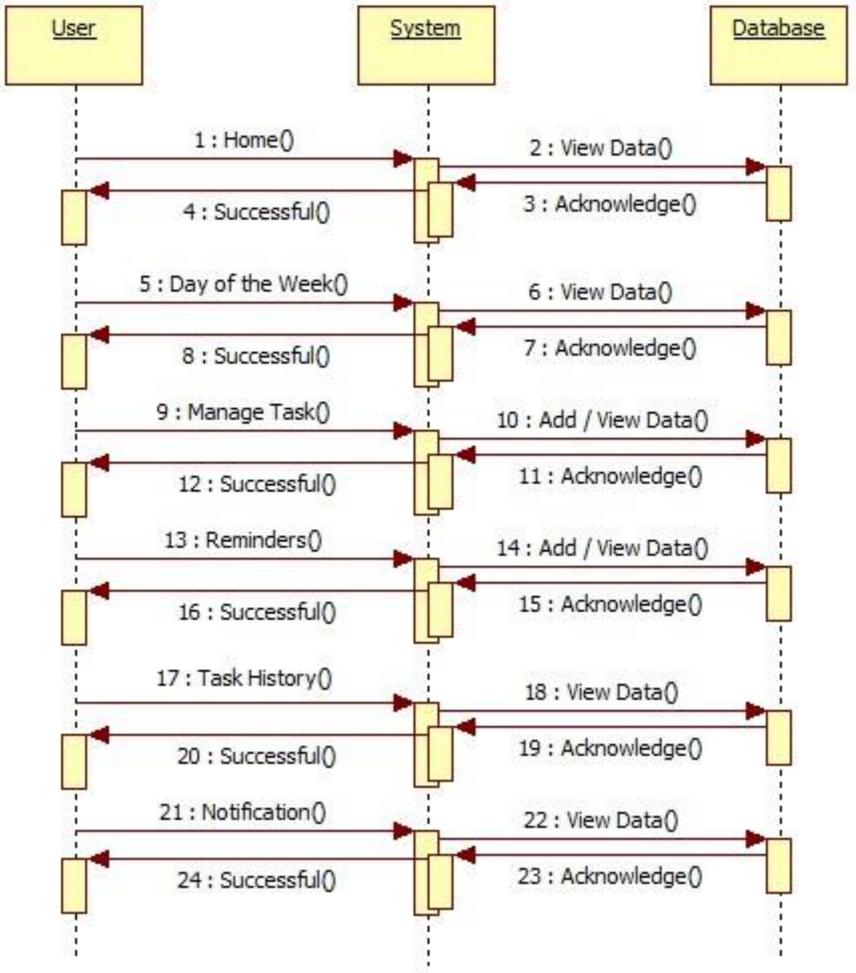


Fig. Sequence Diagram of User

Activity Diagram

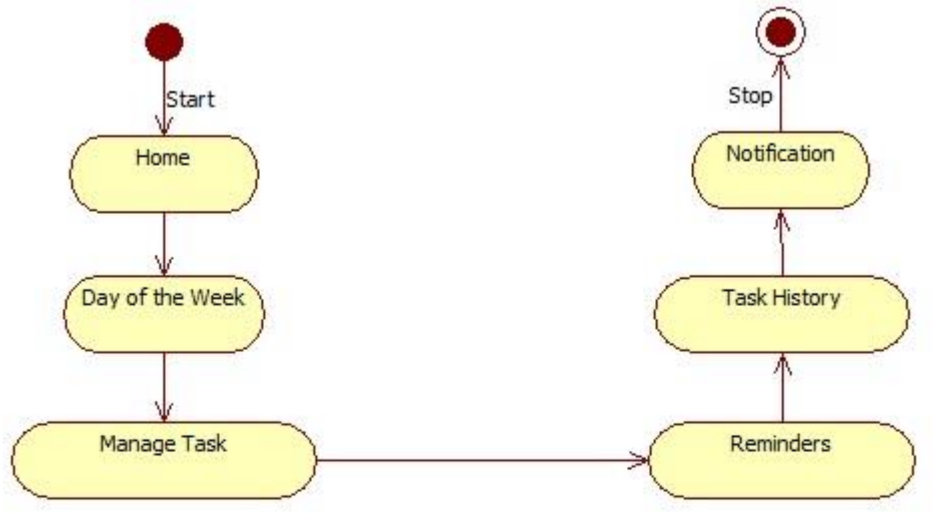
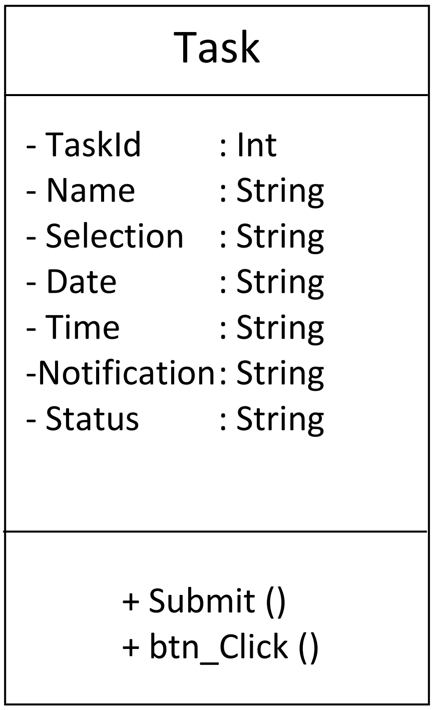


Fig. Activity Diagram of User

Class Diagram



Task

- TaskId : Int
- Name : String
- Selection : String
- Date : String
- Time : String
-Notification: String
- Status : String

+ Submit ()
+ btn_Click ()

Data Flow Diagram (DFD's)

A data flow diagram is graphical tool used to describe and analyze movement of data through a system. These are the central tool and the basis from which the other components are developed. The transformation of data from input to output, through processed, may be described logically and independently of physical components associated with the system. These are known as the logical data flow diagrams. The physical data flow diagrams show the actual implements and movement of data between people, departments and workstations. A full description of a system actually consists of a set of data flow diagrams. Using two familiar notations Yourdon, Gane and Sarson notation develops the data flow diagrams. Each component in a DFD is labeled with a descriptive name. Process is further identified with a number that will be used for identification purpose. The development of DFD's is done in several levels. Each process in lower level diagrams can be broken down into a more detailed DFD in the next level. The lop-level diagram is often called context diagram. It consists a single process bit, which plays vital role in studying the current system. The process in the context level diagram is exploded into other process at the first level DFD.

The idea behind the explosion of a process into more process is that understanding at one level of detail is exploded into greater detail at the next level. This is done until further explosion is necessary and an adequate amount of detail is described for analyst to understand the process.

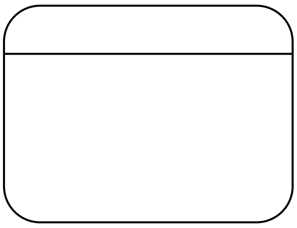
Larry Constantine first developed the DFD as a way of expressing system requirements in a graphical from, this lead to the modular design.

A DFD is also known as a “bubble Chart” has the purpose of clarifying system requirements and identifying major transformations that will become programs in system design. So it is the starting point of the design to the lowest level of detail. A DFD consists of a series of bubbles joined by data flows in the system.

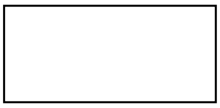
DFD SYMBOLS:

In the DFD, there are four symbols

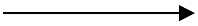
1. A square defines a source(originator) or destination of system data
2. An arrow identifies data flow. It is the pipeline through which the information flows
3. A circle or a bubble represents a process that transforms incoming data flow into outgoing data flows.
4. An open rectangle is a data store, data at rest or a temporary repository of data



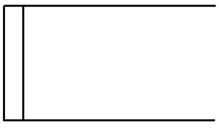
Process that transforms data flow.



Source or Destination of data



Data flow



Data Store

CONSTRUCTING A DFD:

Several rules of thumb are used in drawing DFD's:

1. Process should be named and numbered for an easy reference. Each name should be representative of the process.
2. The direction of flow is from top to bottom and from left to right. Data traditionally flow from source to the destination although they may flow back to the source. One way to indicate this is to draw long flow line back to a source. An alternative way is to repeat the source symbol as a destination. Since it is used more than once in the DFD it is marked with a short diagonal.
3. When a process is exploded into lower level details, they are numbered.
4. The names of data stores and destinations are written in capital letters. Process and dataflow names have the first letter of each work capitalized

A DFD typically shows the minimum contents of data store. Each data store should contain all the data elements that flow in and out.

Questionnaires should contain all the data elements that flow in and out. Missing interfaces redundancies and like is then accounted for often through interviews.

SAILENT FEATURES OF DFD's

1. The DFD shows flow of data, not of control loops and decision are controlled considerations do not appear on a DFD.
2. The DFD does not indicate the time factor involved in any process whether the data flows take place daily, weekly, monthly or yearly.
3. The sequence of events is not brought out on the DFD.

TYPES OF DATA FLOW DIAGRAMS

1. Current Physical
2. Current Logical
3. New Logical
4. New Physical

CURRENT PHYSICAL:

In Current Physical DFD process label include the name of people or their positions or the names of computer systems that might provide some of the overall system-processing label includes an identification of the technology used to process the data. Similarly, data flows and data stores are often labels with the names of the actual physical media on which data are stored such as file folders, computer files, business forms or computer tapes.

CURRENT LOGICAL:

The physical aspects at the system are removed as much as possible so that the current system is reduced to its essence to the data and the processors that transform them regardless of actual physical form.

NEW LOGICAL:

This is exactly like a current logical model if the user were completely happy with the user were completely happy with the functionality of the current system but had problems with how it was implemented typically through the new logical model will differ from current logical model while having additional functions, absolute function removal and inefficient flows recognized.

NEW PHYSICAL:

The new physical represents only the physical implementation of the new system.

RULES GOVERNING THE DFD'S

PROCESS

1) No process can have only outputs.
2) No process can have only inputs. If an object has only inputs than it must be a sink.
3) A process has a verb phrase label.

DATA STORE

1) Data cannot move directly from one data store to another data store, a process must move data.
2) Data cannot move directly from an outside source to a data store, a process, which receives, must move data from the source and place the data into data store
3) A data store has a noun phrase label.

SOURCE OR SINK

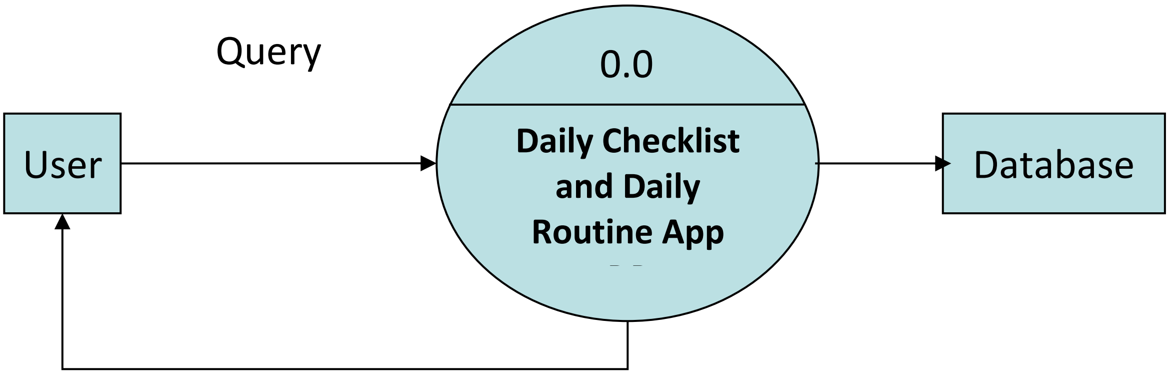
The origin and /or destination of data.

1) Data cannot move direly from a source to sink it must be moved by a process
2) A source and /or sink has a noun phrase land

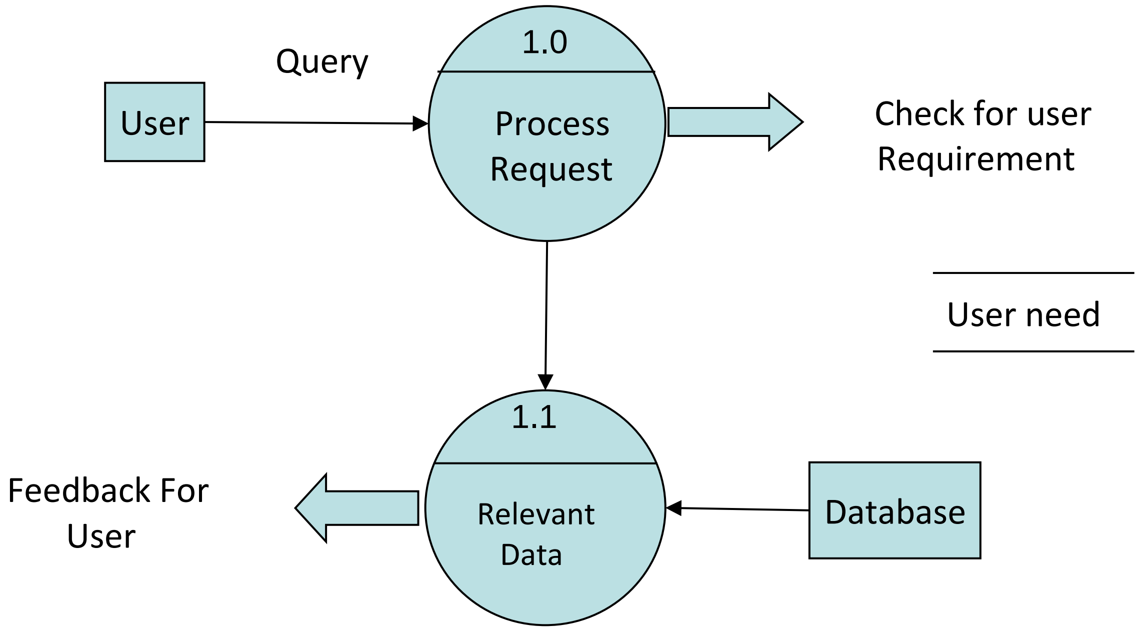
DATA FLOW

1) A Data Flow has only one direction of flow between symbols. It may flow in both directions between a process and a data store to show a read before an update. The later it usually indicated however by two separate arrows since these happen at different type.
2) A join in DFD means that exactly the same data comes from any of two or more different processes data store or sink to a common location.
3) A data flow cannot go directly back to the same process it leads. There must be at least one other process that handles the data flow produce some other data flow returns the original data into the beginning process.
4) A Data flow to a data store means update (delete or change).
5) A data Flow from a data store means retrieve or use.

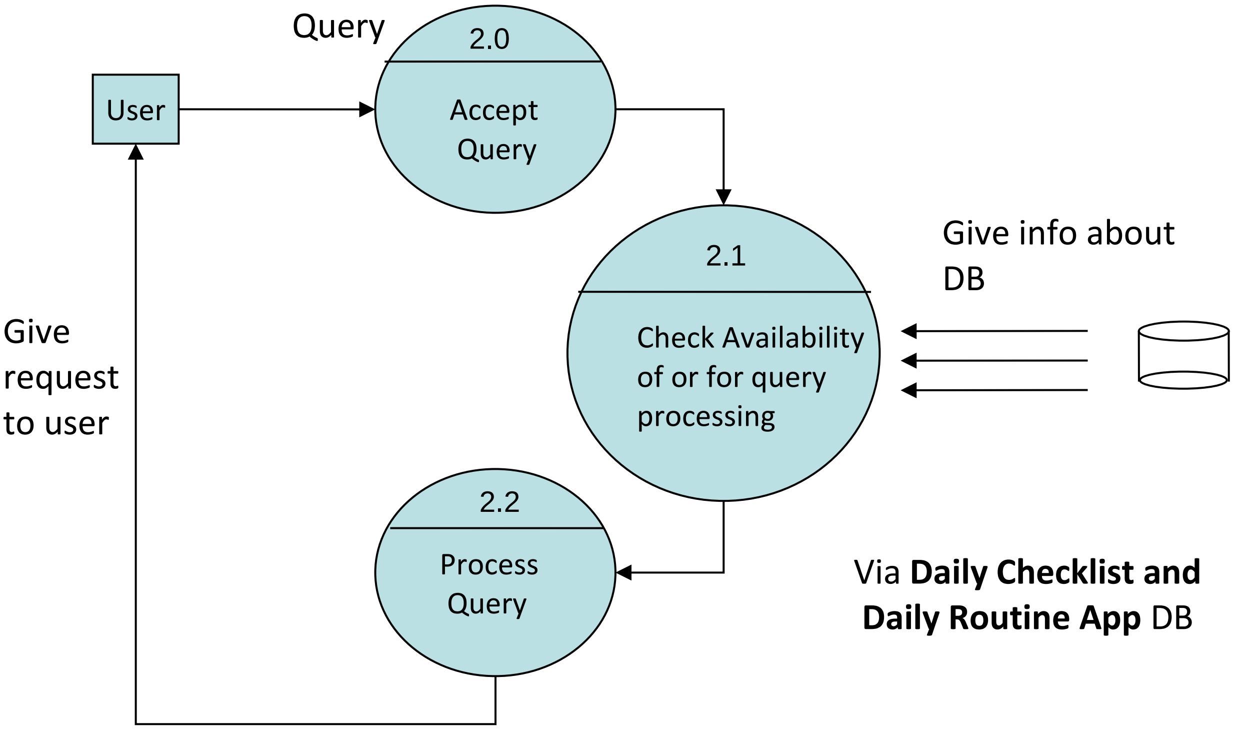
Data Flow Diagrams (DFD’s)



DATABASE DETAIL

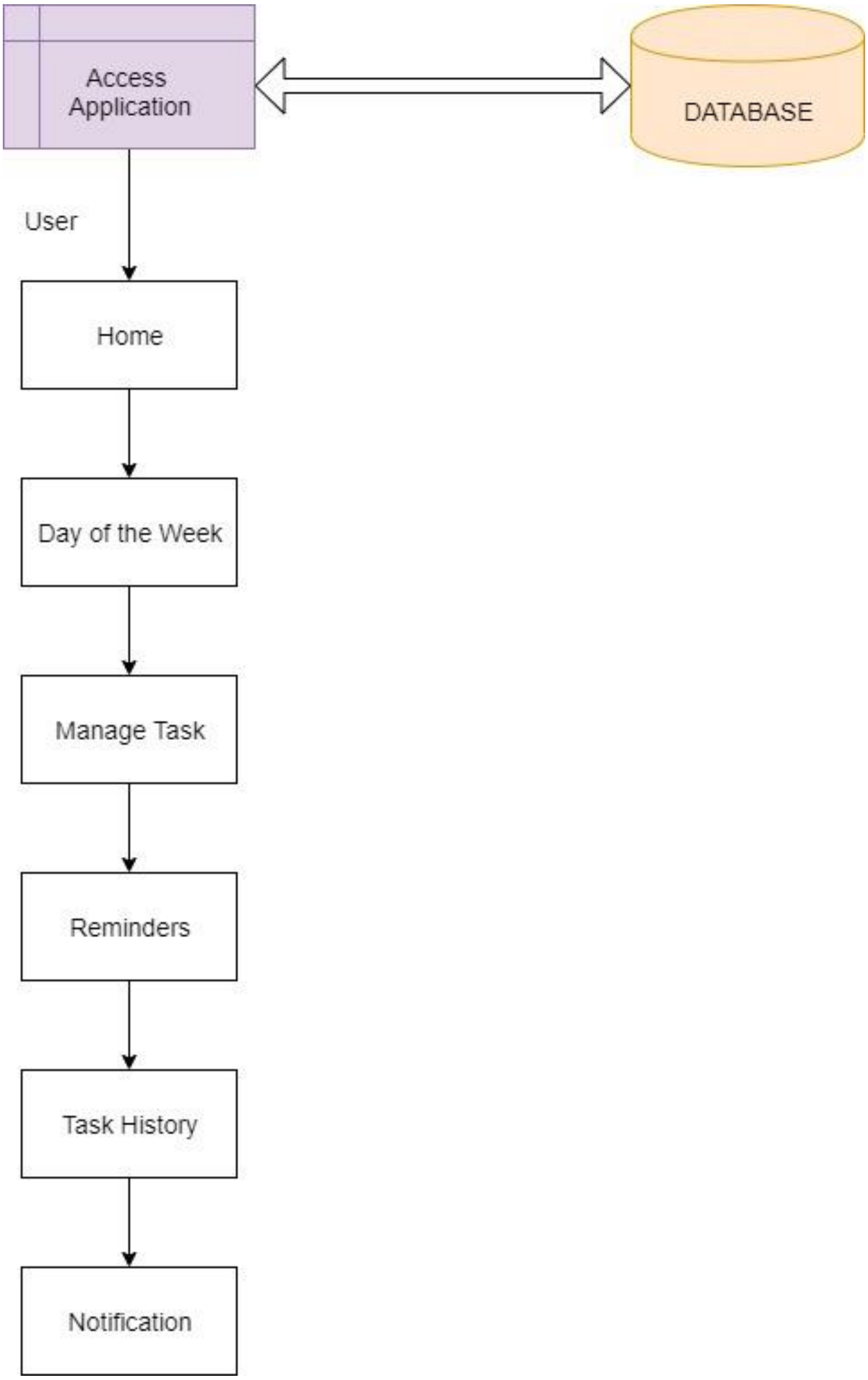

Query
User
1.0
Process Request
Check for user Requirement
User need
1.1
Feedback For User
Relevant Data
Database

LEVEL 1 DFD

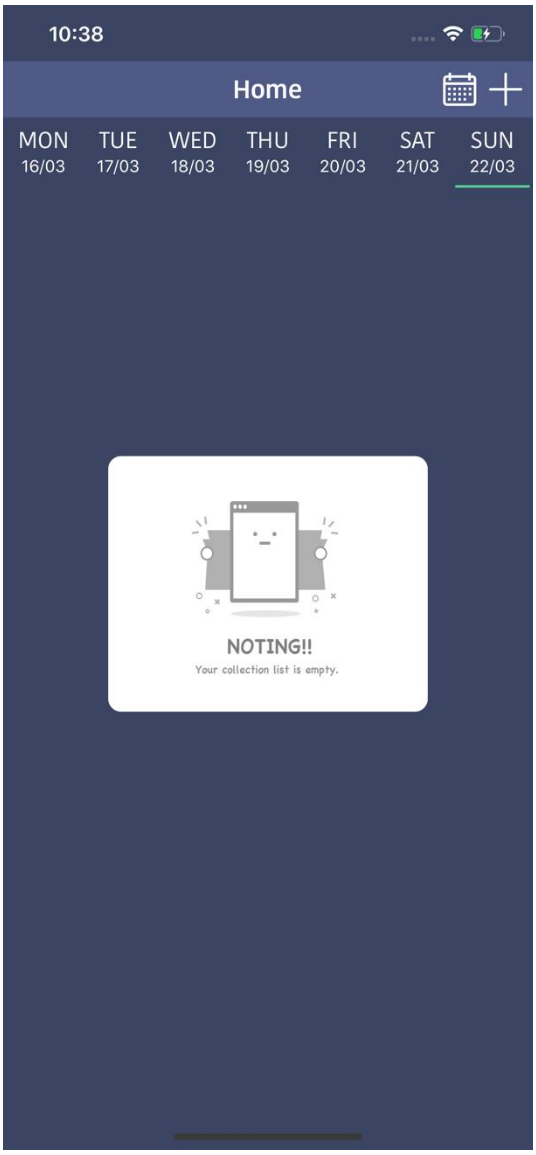

Query
2.0
Accept Query
User
2.1
Check Availability of or for query processing
Give info about DB
Give request to user
2.2
Process Query
Via **Daily Checklist and Daily Routine App** DB

LEVEL 2 DFD: PREDICTION

System Architecture

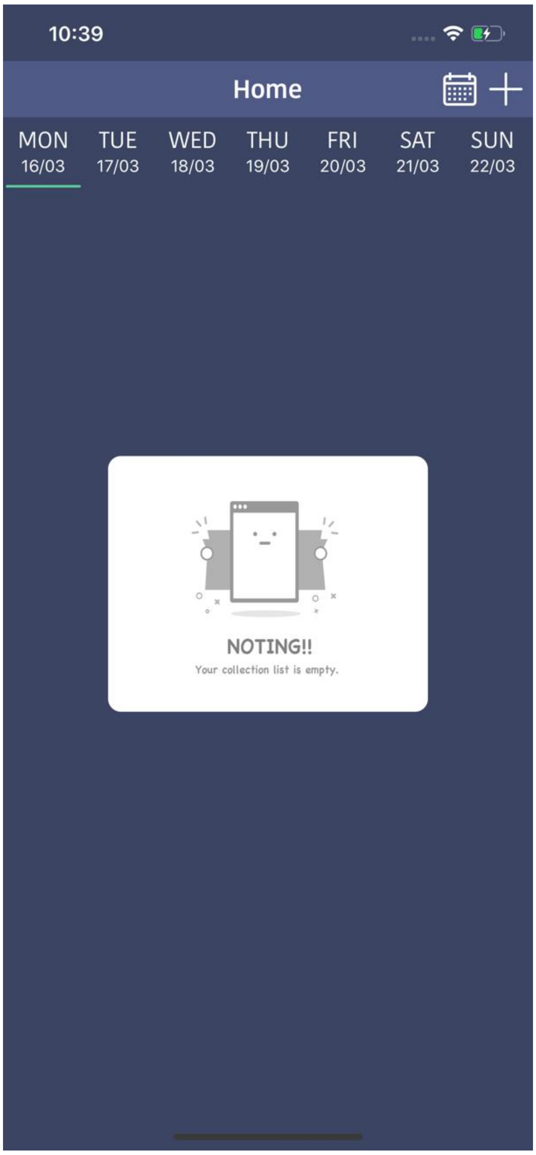


Snapshots



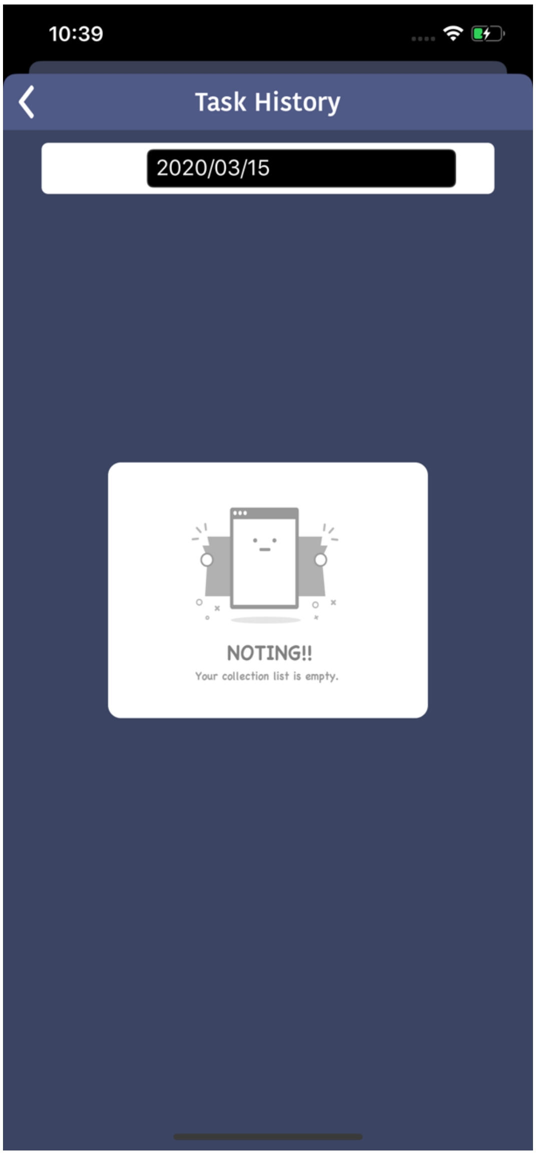
NOTING!!

Your collection list is empty.



NOTING!!

Your collection list is empty.



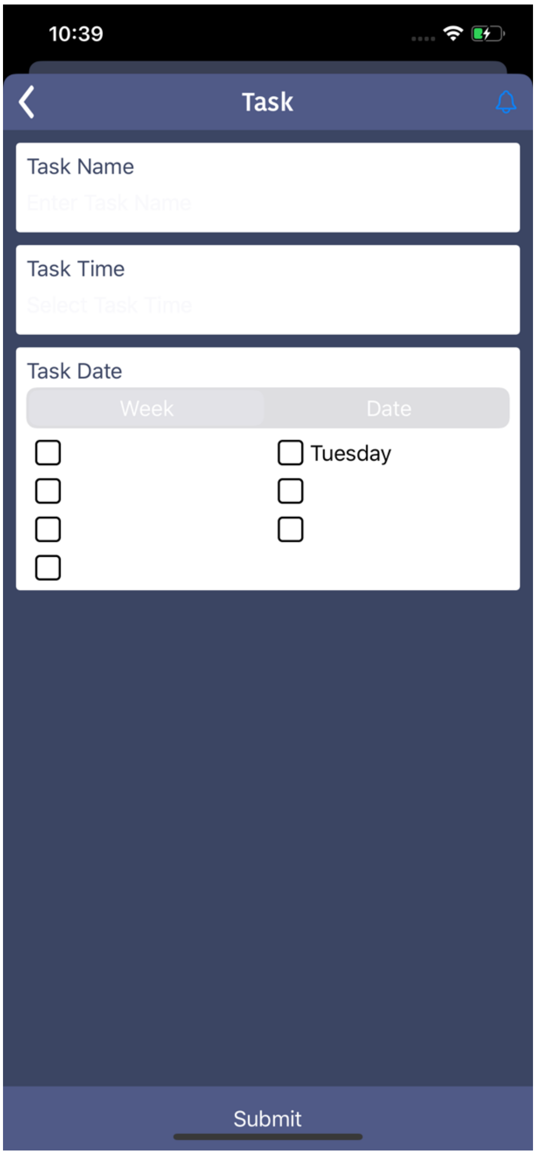
10:39

Task History

2020/03/15

NOTING!!

Your collection list is empty.



10:39

Task

Task Name

Enter Task Name

Task Time

Select Task Time

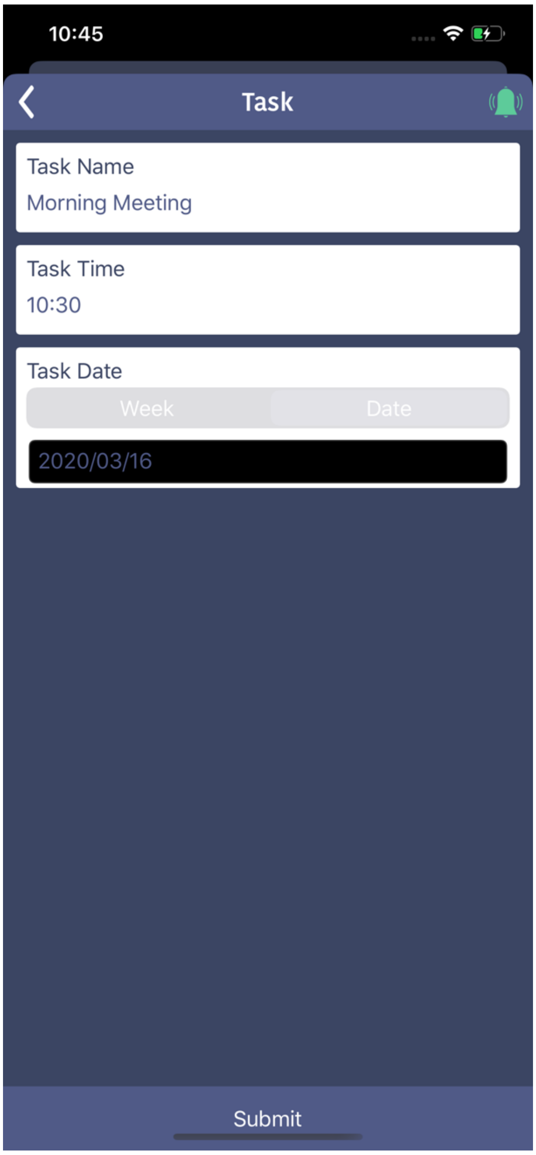
Task Date

Week | Date

□
□
□
□

□ Tuesday
□
□

Submit



10:45

Task

Task Name

Morning Meeting

Task Time

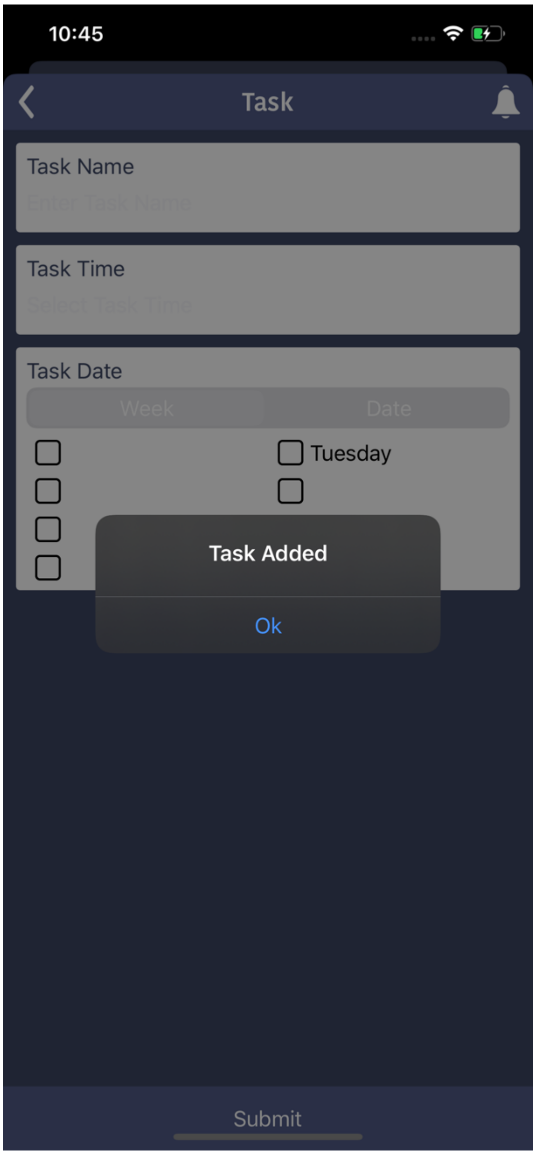
10:30

Task Date

Week | Date

2020/03/16

Submit



10:45

Task

Task Name

Enter Task Name

Task Time

Select Task Time

Task Date

Week | Date

□ □ Tuesday

□ □

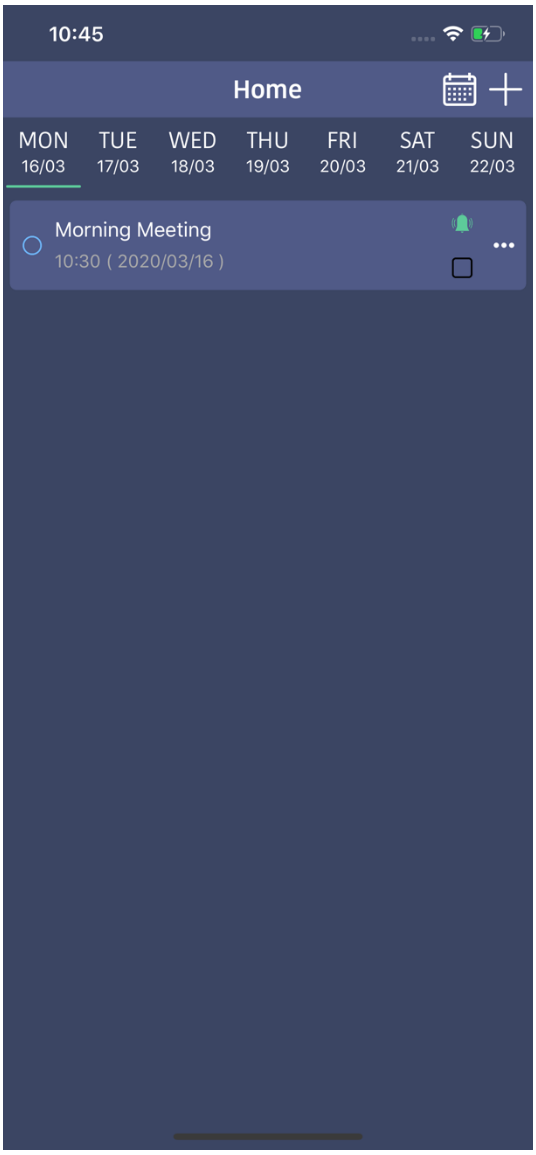
□

□

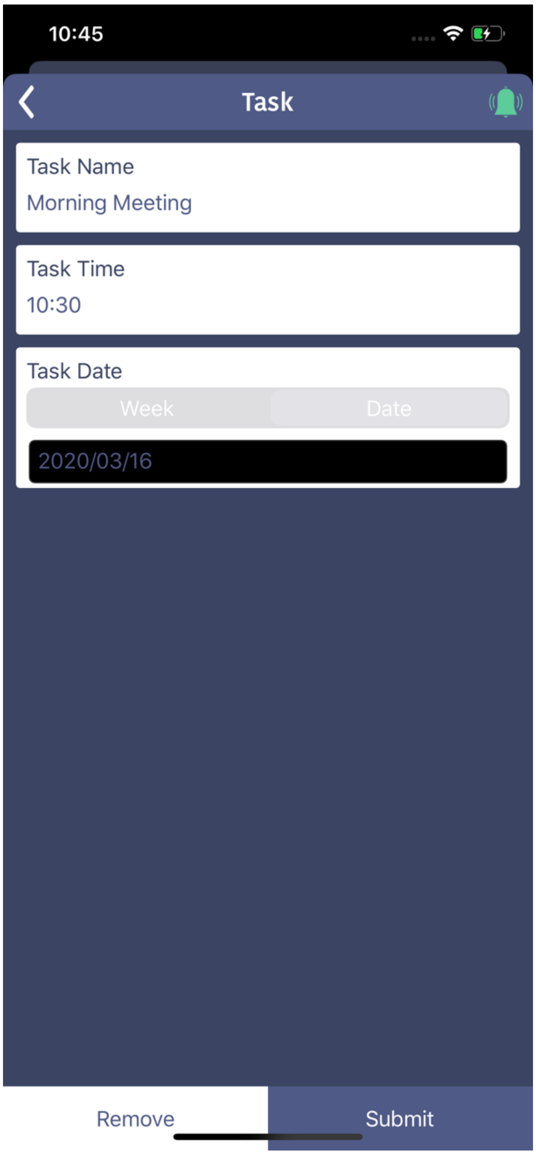
Task Added

Ok

Submit



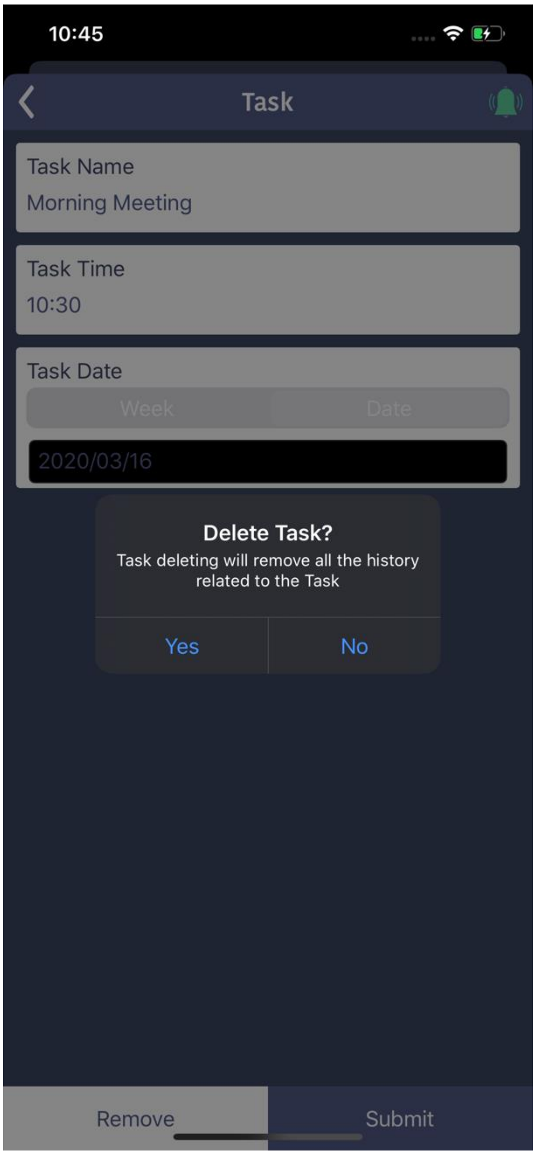
○ Morning Meeting
10:30 (2020/03/16) □



Task Name
Morning Meeting

Task Time
10:30

Task Date
2020/03/16

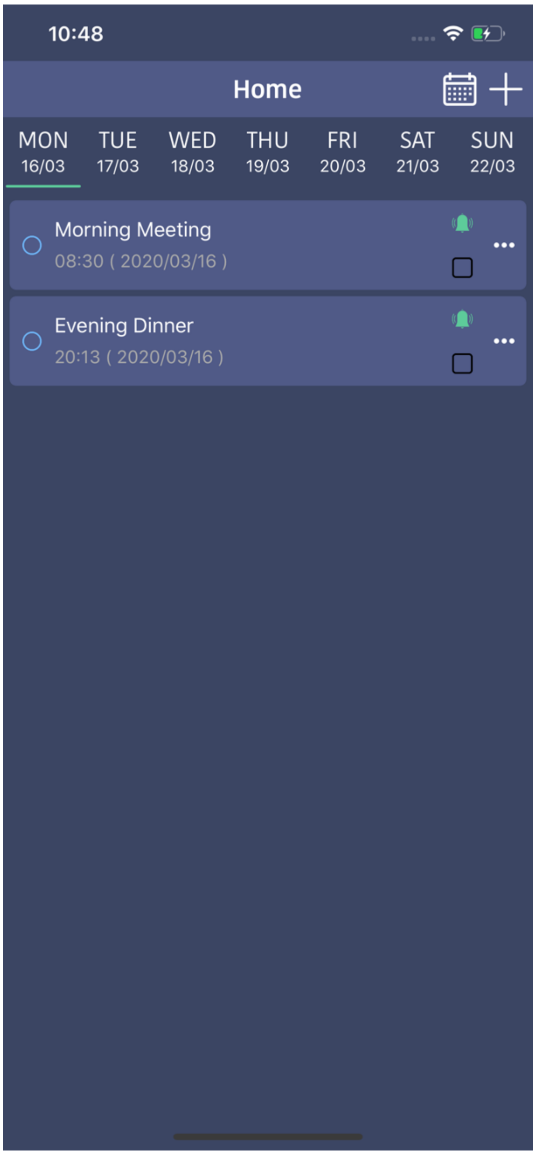


Task Name
Morning Meeting

Task Time
10:30

Task Date
Week | Date
2020/03/16

Delete Task?
Task deleting will remove all the history related to the Task



○ Morning Meeting
08:30 (2020/03/16) □

○ Evening Dinner
20:13 (2020/03/16) □

PROJECT IMPLEMENTATION

Project Implementation Technology

Front End: XCode

Programming Language: objective C

Backend: SQLite

Software Requirements:

System:

- MAC OS 10
- X Code 8 or above
- Safari Browser

iPhone:

- iOS 10

Hardware Requirements:

System:

- Processor: Core i5
- Memory: 4 GB
- HDD: 160 GB

iPhone:

- Memory: 2 GB or more
- Storage: 8 GB or more

OVERVIEW OF TECHNOLOGIES USED

Introduction to iOS

Objective-C is a general-purpose, object-oriented programming language that adds Smalltalk-style messaging to the C programming language. This is the main programming language used by Apple for the OS X and iOS operating systems and their respective APIs, Cocoa and Cocoa Touch.

This reference will take you through simple and practical approach while learning Objective-C Programming language.

The programming language Objective-C was originally developed in the early 1980s. It was selected as the main language used by **NeXT** for its **NeXTSTEP** operating system, from which OS X and iOS are derived. Portable Objective-C programs that do not use the **Cocoa** or **Cocoa Touch** libraries, or those using parts that may be ported or re-implemented for other systems, can also be compiled for any system supported by GNU Compiler Collection (GCC) or Clang. Objective-C source code 'implementation' program files usually have .m filename extensions, while Objective-C 'header/interface' files have .h extensions, the same as C header files. Objective-C++ files are denoted with an .mm file extension.

Object-Oriented Programming

Fully supports object-oriented programming, including the four pillars of object-oriented development:

- Encapsulation
- Data hiding
- Inheritance
- Polymorphism

Foundation Framework

Foundation Framework provides large set of features and they are listed below.

- It includes a list of extended datatypes like NSArray, NSDictionary, NSSet and so on.
- It consists of a rich set of functions manipulating files, strings, etc.
- It provides features for URL handling, utilities like date formatting, data handling, error handling, etc.

Learning Objective-C

The most important thing to do when learning Objective-C is to focus on concepts and not get lost in language technical details.

The purpose of learning a programming language is to become a better programmer; that is, to become more effective at designing and implementing new systems and at maintaining old ones.

Use of Objective-C

Objective-C, as mentioned earlier, is used in iOS and Mac OS X. It has large base of iOS users and largely increasing Mac OS X users. In addition, since Apple focuses on quality first and it’s wonderful for those who started learning Objective-C.

History

Objective-C was created primarily by Brad Cox and Tom Love in the early 1980s at their company StepStone. Both had been introduced to Smalltalk while at ITT Corporation's Programming Technology Center in 1981. The earliest work on Objective-C traces back to around that time. Cox was intrigued by problems of true reusability in software design and programming. He realized that a language like Smalltalk would be invaluable in building development environments for system developers at ITT. However, he and Tom Love also recognized that backward compatibility with C was critically important in ITT's telecom engineering milieu.

Cox began writing a pre-processor for C to add some of the abilities of Smalltalk. He soon had a working implementation of an object-oriented extension to the C language, which he called "OOPC" for Object-Oriented Pre-Compiler. Love was hired by Schlumberger Research in 1982 and had the opportunity to acquire the first commercial copy of Smalltalk-80, which further influenced the development of their brainchild.

In order to demonstrate that real progress could be made, Cox showed that making interchangeable software components really needed only a few practical changes to existing tools. Specifically, they needed to support objects in a flexible manner, come supplied with a usable set of libraries, and allow for the code (and any resources needed by the code) to be bundled into one cross-platform format.

Love and Cox eventually formed a new venture, Productivity Products International (PPI), to commercialize their product, which coupled an Objective-C compiler with class libraries. In 1986, Cox published the main description of Objective-C in its original form in the book Object-Oriented Programming, An Evolutionary Approach. Although he was careful to point out that there is more to the problem of reusability than just the language, Objective-C often found itself compared feature for feature with other languages.

Popularization through NeXT

In 1988, NeXT licensed Objective-C from StepStone (the new name of PPI, the owner of the Objective-C trademark) and extended the GCC compiler to support Objective-C. NeXT developed the AppKit and Foundation Kit libraries on which the NeXTSTEP user interface and Interface Builder were based. While the NeXT workstations failed to make a great impact in the marketplace, the tools were widely lauded in the industry. This led NeXT to drop hardware production and focus on software tools, selling NeXTSTEP (and OpenStep) as a platform for custom programming.

In order to circumvent the terms of the GPL, NeXT had originally intended to ship the Objective-C frontend separately, allowing the user to link it with GCC to produce the compiler executable. After being initially accepted by Richard M. Stallman, this plan was rejected after Stallman consulted with GNU's lawyers and NeXT agreed to make Objective-C part of GCC.

The work to extend GCC was led by Steve Naroff, who joined NeXT from StepStone. The compiler changes were made available as per GPL license terms, but the runtime libraries were not, rendering the open source contribution unusable to the general public. This led to other parties developing such runtime libraries under open source license. Later, Steve Naroff was also principal contributor to work at Apple to build the Objective-C frontend to Clang.

The GNU project started work on its free software implementation of Cocoa, named GNUstep, based on the OpenStep standard. Dennis Glatting wrote the first GNU Objective-C runtime in 1992. The GNU Objective-C runtime, which has been in use since 1993, is the one developed by Kresten Krab Thorup when he was a university student in Denmark. Thorup also worked at NeXT from 1993 to 1996.

Apple development and Swift

After acquiring NeXT in 1996, Apple Computer used OpenStep in its new operating system, OS X. This included Objective-C, NeXT's Objective-C-based developer tool, Project Builder, and its interface design tool, Interface Builder, both now merged into one application, Xcode. Most of Apple's current Cocoa API is based on OpenStep interface objects and is the most significant Objective-C environment being used for active development.

At WWDC 2014, Apple introduced a new language, Swift, which was characterized as "Objective-C without the C".

Syntax

Objective-C is a thin layer atop C, and is a "strict superset" of C, meaning that it is possible to compile any C program with an Objective-C compiler, and to freely include C language code within an Objective-C class.

Objective-C derives its object syntax from Smalltalk. All of the syntax for non-object-oriented operations (including primitive variables, pre-processing, expressions, function declarations, and function calls) are identical to those of C, while the syntax for object-oriented features is an implementation of Smalltalk-style messaging.

Messages

The Objective-C model of object-oriented programming is based on message passing to object instances. In Objective-C, one does not call a method; one sends a message. This is unlike the Simula-style programming model used by C++. The difference between these two concepts is in how the code referenced by the method or message name is executed. In a Simula-style language, the method name is in most cases bound to a section of code in the target class by the compiler. In Smalltalk and Objective-C, the target of a message is resolved at runtime, with the receiving object itself interpreting the message. A method is identified by a selector or SEL a NUL-terminated string representing its name and resolved to a C method pointer implementing it: an IMP. A consequence of this is that the message-passing system has no type checking. The object to which the message is directed the receiver is not guaranteed to respond to a message, and if it does not, it raises an exception.

Sending the message method to the object pointed to by the pointer obj would require the following code in C++.

```
obj->method(argument);
```

In Objective-C, this is written as follows:

```
[obj method:argument];
```

Both styles of programming have their strengths and weaknesses. Object-oriented programming in the Simula (C++) style allows multiple inheritance and faster execution by using compile-time binding whenever possible, but it does not support dynamic binding by default. It also forces all methods to have a corresponding implementation unless they are abstract. The Smalltalk-style programming as used in Objective-C allows messages to go unimplemented, with the method resolved to its implementation at runtime. For example, a message may be sent to a collection of objects, to which only some will be expected to respond, without fear of producing runtime errors. Message passing also does not require that an object be defined at compile time. An implementation is still required for the method to be called in the derived object. (See the dynamic typing section below for more advantages of dynamic (late) binding.)

Interfaces and Implementations

Objective-C requires that the interface and implementation of a class be in separately declared code blocks. By convention, developers place the interface in a header file and the implementation in a code file. The header files, normally suffixed .h, are similar to C header files while the implementation (method) files, normally suffixed .m, can be very similar to C code files.

Interface

In other programming languages, this is called a "class declaration".

The interface of a class is usually defined in a header file. A common convention is to name the header file after the name of the class, e.g. Ball.h would contain the interface for the class Ball.

An interface declaration takes the form:

```
@interface classname : superclassname {
 // instance variables
}
+ classMethod1;
+ (return_type)classMethod2;
+ (return_type)classMethod3:(param1_type)param1_varName;

- (return_type)instanceMethod1With1Parameter:(param1_type)param1_varName;
- (return_type)instanceMethod2With2Parameters:(param1_type)param1_varName
param2_callName:(param2_type)param2_varName;
@end
```

Protocols

In other programming languages, these are called "interfaces".

Objective-C was extended at NeXT to introduce the concept of multiple inheritance of specification, but not implementation, through the introduction of protocols. This is a pattern achievable either as an abstract multiple inherited base class in C++, or as an "interface" (as in Java and C#). Objective-C makes use of ad hoc protocols called informal protocols and compiler-enforced protocols called formal protocols.

An informal protocol is a list of methods that a class can opt to implement. It is specified in the documentation, since it has no presence in the language. Informal protocols are implemented as a category (see below) on NSObject and often include optional methods, which, if implemented, can change the behavior of a class. For example, a text field class might have a delegate that implements an informal protocol with an optional method for performing auto-completion of user-typed text. The text field discovers whether the delegate implements that method (via reflection) and, if so, calls the delegate's method to support the auto-complete feature.

A formal protocol is similar to an interface in Java, C#, and Ada 2005. It is a list of methods that any class can declare itself to implement. Versions of Objective-C before 2.0 required that a class must implement all methods in a protocol it declares itself as adopting; the compiler will emit an error if the class does not implement every method from its declared protocols. Objective-C 2.0 added support for marking certain methods in a protocol optional, and the compiler will not enforce implementation of optional methods.

A class must be declared to implement that protocol to be said to conform to it. This is detectable at runtime. Formal protocols cannot provide any implementations; they simply assure callers that classes that conform to the protocol will provide implementations. In the NeXT/Apple library, protocols are frequently used by the Distributed Objects system to represent the abilities of an object executing on a remote system.

The syntax:

```
@protocol NSLocking
- (void)lock;
- (void)unlock;
@end
```

denotes that there is the abstract idea of locking. By stating in the class definition that the protocol is implemented,

```
@interface NSLock : NSObject <NSLocking>
//...
@end
```

Dynamic typing

Objective-C, like Smalltalk, can use dynamic typing: an object can be sent a message that is not specified in its interface. This can allow for increased flexibility, as it allows an object to "capture" a message and send the message to a different object that can respond to the message appropriately, or likewise send the message on to another object. This behavior is known as message forwarding or delegation (see below). Alternatively, an error handler can be used in case the message cannot be forwarded. If an object does not forward a message, respond to it, or handle an error, then the system will generate a runtime exception. If messages are sent to nil (the null object pointer), they will be silently ignored or raise a generic exception, depending on compiler options.

Static typing information may also optionally be added to variables. This information is then checked at compile time. In the following four statements, increasingly specific type information is provided. The statements are equivalent at runtime, but the extra information allows the compiler to warn the programmer if the passed argument does not match the type specified.

```
- (void)setMyValue:(id)foo;
```

In the above statement, foo may be of any class.

```
- (void)setMyValue:(id<NSCopying>)foo;
```

In the above statement, foo may be an instance of any class that conforms to the `NSCopying` protocol.

```
- (void)setMyValue:(NSNumber *)foo;
```

In the above statement, foo must be an instance of the *NSNumber* class.

```
- (void)setMyValue:(NSNumber<NSCopying> *)foo;
```

In the above statement, foo must be an instance of the NSNumber class, and it must conform to the `NSCopying` protocol.

Language variants

Objective-C++

Objective-C++ is a language variant accepted by the front-end to the GNU Compiler Collection and Clang, which can compile source files that use a combination of C++ and Objective-C syntax. Objective-C++ adds to C++ the extensions that Objective-C adds to C. As nothing is done to unify the semantics behind the various language features, certain restrictions apply:

- A C++ class cannot derive from an Objective-C class and vice versa.
- C++ namespaces cannot be declared inside an Objective-C declaration.
- Objective-C declarations may appear only in global scope, not inside a C++ namespace
- Objective-C classes cannot have instance variables of C++ classes that lack a default constructor or that have one or more virtual methods, but pointers to C++ objects can be used as instance variables without restriction (allocate them with new in the -init method).
- C++ "by value" semantics cannot be applied to Objective-C objects, which are only accessible through pointers.
- An Objective-C declaration cannot be within a C++ template declaration and vice versa. However, Objective-C types (e.g., Classname *) can be used as C++ template parameters.
- Objective-C and C++ exception handling is distinct; the handlers of each cannot handle exceptions of the other type. This is mitigated in recent runtimes as Objective-C exceptions are either replaced by C++ exceptions completely (Apple runtime), or partly when Objective-C++ library is linked (GNUstep libobjc2).

- Care must be taken since the destructor calling conventions of Objective-C and C++'s exception run-time models do not match (i.e., a C++ destructor will not be called when an Objective-C exception exits the C++ object's scope). The new 64-bit runtime resolves this by introducing interoperability with C++ exceptions in this sense.
- Objective-C blocks and C++11 lambdas are distinct entities. However, a block is transparently generated on Mac OS X when passing a lambda where a block is expected.

Library use

Objective-C today is often used in tandem with a fixed library of standard objects (often known as a "kit" or "framework"), such as Cocoa, GNUstep or ObjFW. These libraries often come with the operating system: the GNUstep libraries often come with Linux-based distributions and Cocoa comes with OS X. The programmer is not forced to inherit functionality from the existing base class (NSObject / OFObject). Objective-C allows for the declaration of new root classes that do not inherit any existing functionality. Originally, Objective-C-based programming environments typically offered an Object class as the base class from which almost all other classes inherited. With the introduction of OpenStep, NeXT created a new base class named NSObject, which offered additional features over Object (an emphasis on using object references and reference counting instead of raw pointers, for example). Almost all classes in Cocoa inherit from NSObject.

Not only did the renaming serve to differentiate the new default behavior of classes within the OpenStep API, but it allowed code that used Object—the original base class used on NeXTSTEP (and, more or less, other Objective-C class libraries)—to co-exist in the same runtime with code that used NSObject (with some limitations). The introduction of the two letter prefix also became a simplistic form of namespaces, which Objective-C lacks. Using a prefix to create an informal packaging identifier became an informal coding standard in the Objective-C community, and continues to this day.

More recently, package managers have started appearing, such as CocoaPods, which aims to be both a package manager and a repository of packages. A lot of open-source Objective-C code that was written in the last few years can now be installed using CocoaPods.

Analysis of the language

Objective-C implementations use a thin runtime system written in C, which adds little to the size of the application. In contrast, most object-oriented systems at the time that it was created used large virtual machine runtimes. Programs written in Objective-C tend to be not much larger than the size of their code and that of the libraries (which generally do not need to be included in the software distribution), in contrast to Smalltalk systems where a large amount of memory was used just to open a window. Objective-C applications tend to be larger than similar C or C++ applications because Objective-C dynamic typing does not allow methods to be stripped or inlined. Since the programmer has such freedom to delegate, forward calls, build selectors on the fly and pass them to the runtime system, the Objective-C compiler cannot assume it is safe to remove unused methods or to inline calls.

Likewise, the language can be implemented atop extant C compilers (in GCC, first as a preprocessor, then as a module) rather than as a new compiler. This allows Objective-C to leverage the huge existing collection of C code, libraries, tools, etc. Existing C libraries can be wrapped in Objective-C wrappers to provide an OO-style interface. In this aspect, it is similar to GObject library and Vala language, which are widely used in development of GTK applications.

All of these practical changes lowered the barrier to entry, likely the biggest problem for the widespread acceptance of Smalltalk in the 1980s.

A common criticism is that Objective-C does not have language support for namespaces. Instead, programmers are forced to add prefixes to their class names, which are traditionally shorter than namespace names and thus more prone to collisions. As of 2007, all Mac OS X classes and functions in the Cocoa programming environment are prefixed with "NS" (e.g. NSObject, NSButton) to identify them as belonging to the Mac OS X or iOS core; the "NS" derives from the names of the classes as defined during the development of NeXTSTEP.

Since Objective-C is a strict superset of C, it does not treat C primitive types as first-class objects.

Unlike C++, Objective-C does not support operator overloading. Also unlike C++, Objective-C allows an object to directly inherit only from one class (forbidding multiple inheritance). However, in most cases, categories and protocols may be used as alternative ways to achieve the same results.

Because Objective-C uses dynamic runtime typing and because all method calls are function calls (or, in some cases, syscalls), many common performance optimizations cannot be applied to Objective-C methods (for example: inlining, constant propagation, interprocedural optimizations, and scalar replacement of aggregates). This limits the performance of Objective-C abstractions relative to similar abstractions in languages such as C++ where such optimizations are possible.

Memory management

The first versions of Objective-C did not support garbage collection. At the time this decision was a matter of some debate, and many people considered long "dead times" (when Smalltalk performed collection) to render the entire system unusable. Some 3rd party implementations have added this feature (most notably GNUstep) and Apple has implemented it as of Mac OS X v10.5. However, in more recent versions of Mac OS X and iOS, garbage collection has been deprecated in favor of Automatic Reference Counting (ARC), introduced in 2011.

With ARC, the compiler inserts retain and release calls automatically into Objective-C code based on static code analysis. The automation relieves the programmer of having to write in memory management code. ARC also adds weak references to the Objective-C language.

Philosophical differences between Objective-C and C++

The design and implementation of C++ and Objective-C represent fundamentally different approaches to extending C.

In addition to C's style of procedural programming, C++ directly supports certain forms of object-oriented programming, generic programming, and metaprogramming. C++ also comes with a large standard library that includes several container classes. Similarly, Objective-C adds object-oriented programming, dynamic typing, and reflection to C. Objective-C does not provide a standard library per se, but in most places where Objective-C is used, it is used with an OpenStep-like library such as OPENSTEP, Cocoa, or GNUstep, which provides functionality similar to C++'s standard library.

One notable difference is that Objective-C provides runtime support for reflective features, whereas C++ adds only a small amount of runtime support to C. In Objective-C, an object can be queried about its own properties, e.g., whether it will respond to a certain message. In C++, this is not possible without the use of external libraries.

The use of reflection is part of the wider distinction between dynamic (run-time) features and static (compile-time) features of a language. Although Objective-C and C++ each employ a mix of both features, Objective-C is decidedly geared toward run-time decisions while C++ is geared toward compile-time decisions. The tension between dynamic and static programming involves many of the classic trade-offs in programming: dynamic features add flexibility, static features add speed and type checking.

Generic programming and metaprogramming can be implemented in both languages using runtime polymorphism. In C++ this takes the form of virtual functions and runtime type identification, while Objective-C offers dynamic typing and reflection. Objective-C lacks compile-time polymorphism (generic functions) entirely, while C++ supports it via function overloading and templates.

Coding

```
//  main.m

#import <UIKit/UIKit.h>
#import "AppDelegate.h"

int main(int argc, char * argv[]) {
    @autoreleasepool {
        return UIApplicationMain(argc, argv, nil, NSStringFromClass([AppDelegate class]));
    }
}
```

FEASIBILITY REPORT

Feasibility Study is a high level capsule version of the entire process intended to answer a number of questions like: What is the problem? Is there any feasible solution to the given problem? Is the problem even worth solving? Feasibility study is conducted once the problem clearly understood. Feasibility study is necessary to determine that the proposed system is Feasible by considering the technical, Operational, and Economical factors. By having a detailed feasibility study the management will have a clear-cut view of the proposed system.

The following feasibilities are considered for the project in order to ensure that the project is variable and it does not have any major obstructions. Feasibility study encompasses the following things:

- Technical Feasibility
- Economic Feasibility
- Operational Feasibility

In this phase, we study the feasibility of all proposed systems, and pick the best feasible solution for the problem. The feasibility is studied based on three main factors as follows.

❖ Technical Feasibility

In this step, we verify whether the proposed systems are technically feasible or not. i.e., all the technologies required to develop the system are available readily or not.

Technical Feasibility determines whether the organization has the technology and skills necessary to carry out the project and how this should be obtained. The system can be feasible because of the following grounds:

- All necessary technology exists to develop the system.
- This system is too flexible and it can be expanded further.
- This system can give guarantees of accuracy, ease of use, reliability and the data security.
- This system can give instant response to inquire.

Our project is technically feasible because, all the technology needed for our project is readily available.

Operating System	: MAC OS 10 or higher, & iOS v10 or Higher (for iOS Devices)
Languages	: Objective C
Database System	: SQL Lite
Documentation Tool	: MS - Word 2016

❖ Economic Feasibility

Economically, this project is completely feasible because it requires no extra financial investment and with respect to time, it’s completely possible to complete this project in 6 months.

In this step, we verify which proposal is more economical. We compare the financial benefits of the new system with the investment. The new system is economically feasible only when the financial benefits are more than the investments and expenditure. Economic Feasibility determines whether the project goal can be within the resource limits allocated to it or not. It must determine whether it is worthwhile to process with the entire project or whether the benefits obtained from the new system are not worth the costs. Financial benefits must be equal or exceed the costs. In this issue, we should consider:

- The cost to conduct a full system investigation.
- The cost of h/w and s/w for the class of application being considered.
- The development tool.
- The cost of maintenance etc...

Our project is economically feasible because the cost of development is very minimal when compared to financial benefits of the application.

❖ Operational Feasibility

In this step, we verify different operational factors of the proposed systems like man-power, time etc., whichever solution uses less operational resources, is the best operationally feasible solution. The solution should also be operationally possible to implement. Operational Feasibility determines if the proposed system satisfied user objectives could be fitted into the current system operation.

- The methods of processing and presentation are completely accepted by the clients since they can meet all user requirements.
- The clients have been involved in the planning and development of the system.
- The proposed system will not cause any problem under any circumstances.

Our project is operationally feasible because the time requirements and personnel requirements are satisfied. We are a team of four members and we worked on this project for three working months.

TESTING

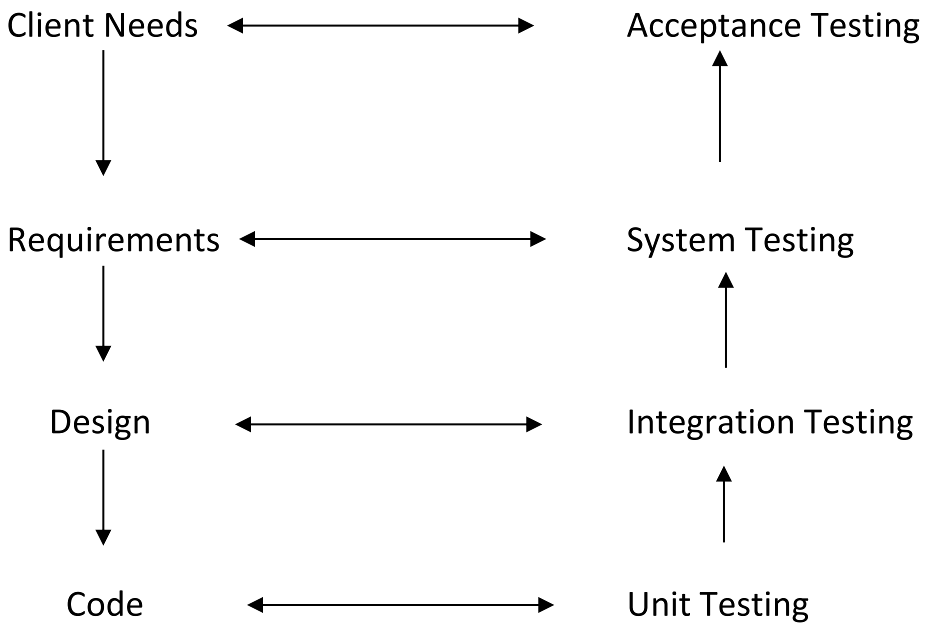
As the project is on bit large scale, we always need testing to make it successful. If each component work properly in all respect and gives desired output for all kind of inputs then project is said to be successful. So, the conclusion is-to make the project successful, it needs to be tested.

The testing done here was System Testing checking whether the user requirements were satisfied. The code for the new system has been written completely using Objective C as the coding language, X Code as the interface for front-end designing. The new system has been tested well with the help of the users and all the applications have been verified from every nook and corner of the user.

Although some applications were found to be erroneous these applications have been corrected before being implemented. The flow of the forms has been found to be very much in accordance with the actual flow of data.

Levels of Testing

In order to uncover the errors present in different phases we have the concept of levels of testing. The basic levels of testing are:



A series of testing is done for the proposed system before the system is ready for the user acceptance testing.

The steps involved in Testing are:

✓ Unit Testing

Unit testing focuses verification efforts on the smallest unit of the software design, the module. This is also known as “Module Testing”. The modules are tested separately. This testing carried out during programming stage itself. In this testing each module is found to be working satisfactorily as regards to the expected output from the module.

✓ Integration Testing

Data can be grossed across an interface; one module can have adverse efforts on another. Integration testing is systematic testing for construction the program structure while at the same time conducting tests to uncover errors associated with in the interface. The objective is to take unit tested modules and build a program structure. All the modules are combined and tested as a whole. Here correction is difficult because the isolation of cause is complicate by the vast expense of the entire program. Thus in the integration testing stop, all the errors uncovered are corrected for the text testing steps.

✓ System testing

System testing is the stage of implementation that is aimed at ensuring that the system works accurately and efficiently for live operation commences. Testing is vital to the success of the system. System testing makes a logical assumption that if all the parts of the system are correct, then goal will be successfully achieved.

✓ Validation Testing

At the conclusion of integration testing software is completely assembled as a package, interfacing errors have been uncovered and corrected and a final series of software tests begins, validation test begins. Validation test can be defined in many ways. But the simple definition is that validation succeeds when the software function in a manner that can reasonably expected by the customer. After validation test has been conducted one of two possible conditions exists.

One is the function or performance characteristics confirm to specifications and are accepted and the other is deviation from specification is uncovered and a deficiency list is created. Proposed system under consideration has been tested by using validation testing and found to be working satisfactorily.

✓ Output Testing

After performing validation testing, the next step is output testing of the proposed system since no system could be useful if it does not produce the required output in the specified format. Asking the users about the format required by them tests the outputs generated by the system under consideration. Here the output format is considered in two ways, one is on the screen and other is the printed format. The output format on the screen is found to be correct as the format was designed in the system designed phase according to the user needs.

For the hard copy also the output comes as the specified requirements by the users. Hence output testing does not result any corrections in the system.

✓ User Acceptance Testing

User acceptance of a system is the key factor of the success of any system. The system under study is tested for the user acceptance by constantly keeping in touch with the prospective system users at the time of developing and making changes wherever required.

Test Cases

VALIDATION CRITERIA

1. In each form, no field which is not null able should be left blank.
2. All numeric fields should be checked for non-numeric values. Similarly, text fields like names should not contain any numeric characters.
3. All primary keys should be automatically generated to prevent the user from entering any existing key.
4. Use of error handling for each Save, Edit, delete and other important operations.
5. Whenever the user Tabs out or Enter from a text box, the data should be validated and if it is invalid, focus should again be sent to the text box with proper message.

ADVANTAGES OF PROJECT

Advantages

- Saves time.
- Helps in achieving goal.
- Helps in planning and executing daily tasks.

Limitations

- **User need to put correct data or else it behaves abnormally.**

Features

1) Load Balancing:

Since the system will be available only the admin logs in the amount of load on server will be limited to time period of admin access.

2) Easy Accessibility:

Records can be easily accessed and store and other information respectively.

3) User Friendly:

The website/application will be giving a very user-friendly approach for all user.

4) Efficient and reliable:

Maintaining the all secured and database on the server which will be accessible according the user requirement without any maintenance cost will be a very efficient as compared to storing all the customer data on the spreadsheet or in physically in the record books.

5) Easy maintenance:

Daily Checklist and Daily Routine App is design as easy way. So maintenance is also easy.

CONCLUSION

This was our project of System Design about “**Daily Checklist and Daily Routine App**” developed in IOS. The Development of this system takes a lot of efforts from us. We think this system gave a lot of satisfaction to all of us. Though every task is never said to be perfect in this development field even more improvement may be possible in this application. We learned so many things and gained a lot of knowledge about development field. We hope this will prove fruitful to us.

BIBLIOGRAPHY

► **Websites**

- ✓ en.wikipedia.org
- ✓ https://shsu-ir.tdl.org/shsu-ir/bitstream/handle/20.500.11875/1164/0781.pdf?sequence=1
- ✓ https://ieeexplore.ieee.org/document/6208293/
- ✓ https://ieeexplore.ieee.org/document/4679917/