**Task Management System**

**Technical Specification Document**

**Table of Contents**

Contents

[1 Introduction……………………………………………………………………………………………………………………………3](#_Toc409649762)

[2 Design Overview……………………………………………………………………………………………………………………4](#_Toc409649763)

[3 Topology Diagram………………………………………………………………………………………………………………….6](#_Toc409649764)

[4 Application Architecture…………………………………………………………………………………………………………7](#_Toc409649765)

[5 Database Architecture…………………………………………………………………………………………………………...14](#_Toc409649766)

|  |  |
| --- | --- |
| 1 Introduction |  |
|  |

**1.1** **Purpose**

The purpose of this document is to outline the technical design of the Task Management Application and provide an overview for the implementation.

Its main purpose is to -

Provide the link between the Functional Specification and the detailed Technical Design documents

Detail the functionality which will be provided by each component or group of components and show how the various components interact in the design

Provide a basis for the Task Management detailed design and development

As is true with any high level design, this document will be updated and refined based on changing requirements.

**1.2** **Scope**

This application will be used by anyone who wants their activity to be tracked down and get alerted so that they don’t miss it. This application can be used by a student who wants to get alerted for exam timing and their study schedule also their sessions. This can be used by businessman to set alarm for meeting. This project uses Angular JS, Node JS, Design patterns and MySql integration in a project implementation. Functionality is kept to bare minimum to avoid complexity.

**1.3** **Audience**

The intended audience for this document is any person who wants keep track of some activity and get alerted when activity time occurs.

|  |  |
| --- | --- |
| 2 Design Overview |  |
|  |

**2.1** **Approach**

This document might be extended in multiple phases over the course of the project -

*Requirements* *Phase* - During the Requirements Phase, the initial version of this document is created, describing the candidate architecture to be validated in the System Design Phase.

*System* *Design* *Phase* - During the System Design phase, the Evolutionary Prototype is created and this document is finalized by establishing a sound architectural foundation for the Construction Phase.

*Construction* *Phase* – During the Construction Phase, this document is not expected to change radically; it is mainly updated to reflect changes in any interface definitions.

*Transition* */* *Training* *Phase* – During the Transition/Training Phase, no further additions or modifications are made to this document. If a new functionality to be implemented it will pass though all the above phases and this document will be updated accordingly.

**2.2** **Architectural** **Goals** **and** **Constraints**

A key Architectural goal is to leverage industry best practices for designing and developing a scalable, enterprise-wide web application.

This application can be used to keep track on activities and maintaining complete task list. This also tells how to design and code a simple web application using javascript technology AngularJS, NodeJS, and MySql.

**2.3** **Guiding** **Principles**

Guiding principles provide a foundation upon which to develop the target architecture for the application, in part by setting the standards and measures that the tool must satisfy. These in turn drive design principles that can be used to validate the design and ensure that it is aligned with Design Principles and Standards.

This application is designed to be flexible. Flexibility is the ability of the application to adapt and evolve to accommodate new requirements without affecting the existing operations. This relies on a modular architecture, which isolates the complexity of integration, presentation, and business logic from each other in order to allow for the easy integration of new technologies and processes within the application.

**2.4** **Design** **Patterns**

Design patterns are elements of reusable object oriented software. A design pattern catalogue is a repository of design patterns. Use of such patterns makes the design of an application transparent. These

patterns have been used successfully by developers in their respective fields, and therefore, the pros and cons of the pattern (as well as implementation issues) are known beforehand. All design patterns are reusable and can be adapted to particular contexts.

Some of the design patterns which will be used in the design and development of this Application are -

Front Controller

Business Delegate

Data Access Object

Façade Pattern

A few more patterns will be added to this list in different phases of the project.

**2.4.1** **Front** **Controller**

The Front Controller pattern helps to implement a centralized entry point that controls and manages user (screen) request handling. The controller manages the handling of the request, including invoking security services such as authentication and authorization, delegating business processing, managing the choice of an appropriate view, handling errors, and managing the selection of content creation strategies.

**2.4.2** **Business** **Delegate**

The Business Delegate pattern helps to reduce coupling between presentation-tier clients and business services. The Business Delegate hides the underlying implementation details of the business service. Methods write in controller act as business delegates.

**2.4.3** **Data** **Access** **Object**

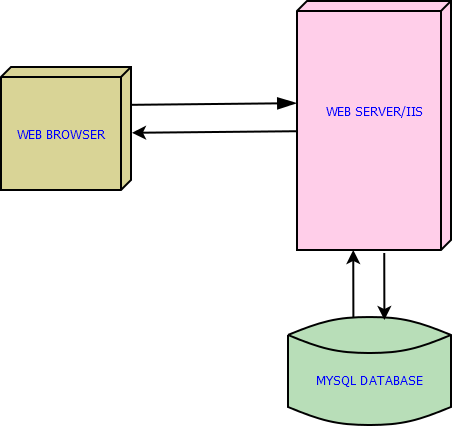
The Data Access Object pattern helps to decouple the service layer from the database thus increasing the portability of the application. . In Node JS we creating repository to deals with data access objects.

* + 1. **Façade Pattern**

Facade pattern hides the complexities of the system and provides an interface to the client using which the client can access the system. Facade Provide a unified interface to a set of interfaces in a subsystem. Facade defines a higher-level interface that makes the subsystem easier to use. In NodeJS we are creating controller object in main file and passing express module object to them.

|  |  |
| --- | --- |
| 3 Topology Diagram |  |
|  |

The diagram below provides a illustration of the System Architecture along with various system components that will be used in architecting the Task Management Application –



Interaction of software components along with its responsibilities is explained below -

**IIS/ Web** **Server** – IIS is used as web as well as application server and is responsible for serving web pages, mostly HTML pages, via the HTTP protocol to clients. The server sends out web pages in response to requests from browsers. A page request is generated when a client clicks a link on a web page in the browser.

The server hosts the business logic and the business model classes of application as well. It serves requests for dynamic HTTP web pages from client.

**MySql Database** – DB stores user profile, credentials and task and subtask data.

**HTTP** - Hyper Text Transport Protocol is the communication protocol used to connect to servers on the

World Wide Web. The primary function of HTTP is to establish a connection with a Web server and transmit HTML pages to the user's browser.

|  |  |
| --- | --- |
| 4 Application Architecture |  |
|  |

Application architecture defines the various components and their interactions in context of a whole system.

At a *conceptual* level, they represent distinct and cohesive aggregations of functionality. This design is based on a tiered approach. “A tier is a logical partition of the separation of concerns of the system. Each tier is assigned its unique responsibility in the system. We view each tier as logically separated from one another. Each tier is loosely coupled with the adjacent tier.” Spring Integration is used in Business and Data access layer. This Application architecture can be represented in the following layers illustrated by the diagram below:

C:\Users\KIRAN\Downloads\ArchitectureDiagramNew.png

4.1 **Presentation** **Layer**

The Client Tier represents the point at which data is consumed by the system’s users which include online users as well as external systems.

A standard Internet Browser such as Chrome is the primary client for the Task Management System Application. Presentation or view layer comprised of HTML + standard tags and angularJS Controller. Controller on user request invokes appropriate service such as user service, task/subtask service. Data-to-content conversion and Content-to-data conversion are the two primary responsibilities of this layer.

**4.2** **Business** **Layer**

The Business layer will implement the business rules for the application. Different components with their responsibilities are:

4.2.1 Controllers: Controller act as an adapter between business logic and presentation layer. It invokes appropriate service for insert, get, delete, update data and act as a business delegate.

4.2.2 Services: Services will be created per BO for providing interface for insert, get, delete, update operations. It is an interface for all business/data services for clients like a web client.

4.2.3 Business Objects: a Business Object will be created per entity like User, Task and SubTask. Each BO has Respository(DAO) injected which is used for data access. All business related rules will be executed in this layer including business validations.

**4.3** **Data** **Access** **Layer**

Each Repository has appropriate URL which is used for data access from DB.

**4.4** **Resource** **Layer**

The resource layer includes the underlying resources that the application uses to deliver its functionality. This is MySql Database to persist information.

**4.5** **Use Case Diagram**

C:\Users\KIRAN\Downloads\use-case1.png

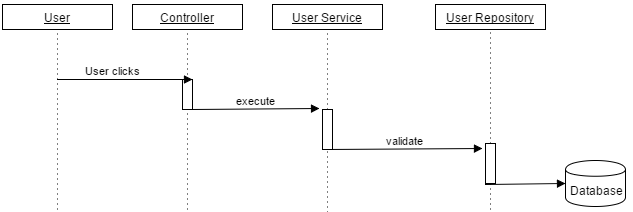
C:\Users\KIRAN\Downloads\use-case2.png

C:\Users\KIRAN\Downloads\use-case3.png

**4.6** **Sequence** **Diagram**

In this section we detail the significant interactions between the major components for the Task Management System Application from a user’s click to executing business logic. Below is a high level logical sequence diagram depicting the significant interactions within the application.

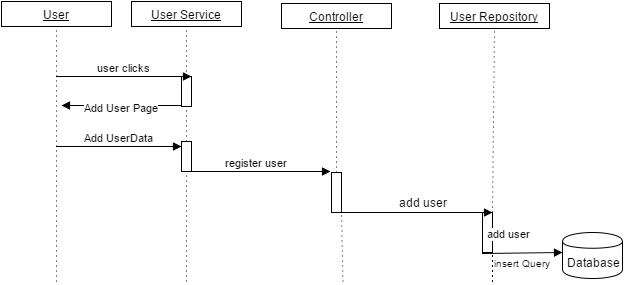
4.6.1 User Login



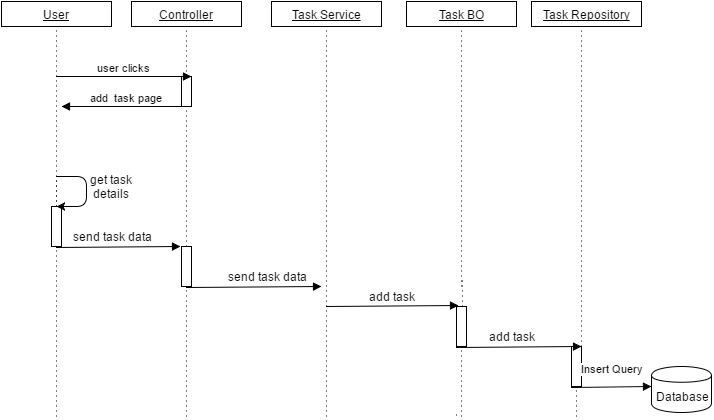
4.6.2 Forgot Password

C:\Users\KIRAN\Downloads\forget password sequence diagramNew.png

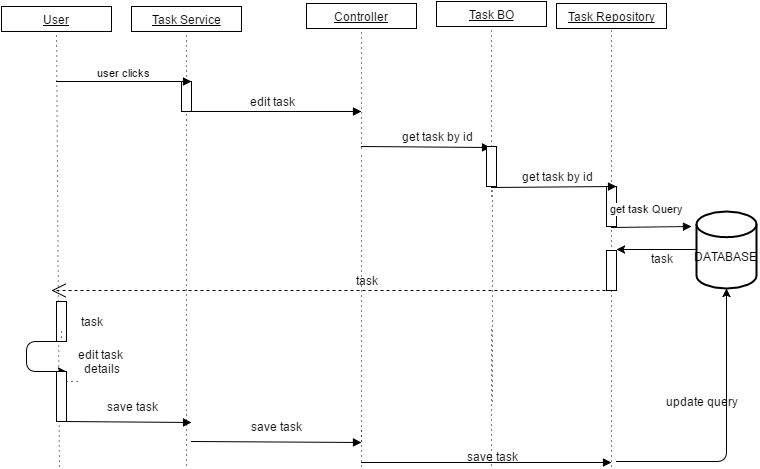
4.6.3 Register User



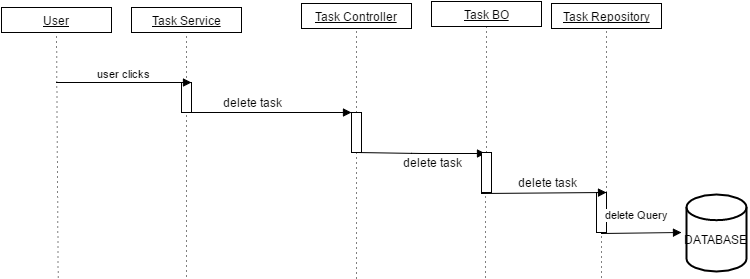
4.6.4 Add New Task



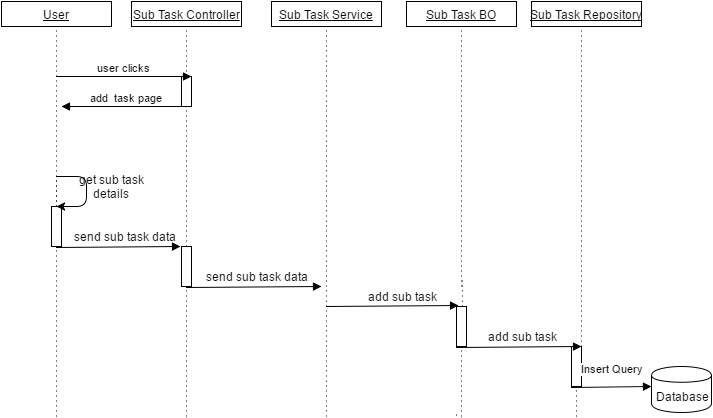
4.6.5 Edit Task



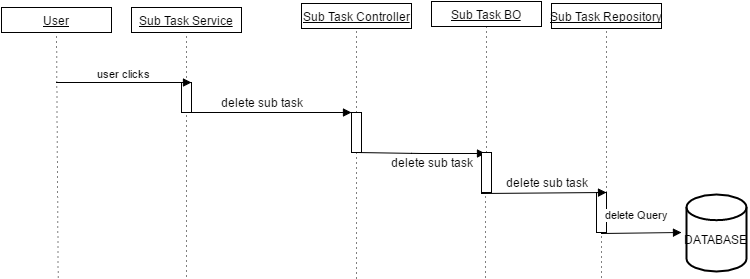
4.6. 6 Delete Task



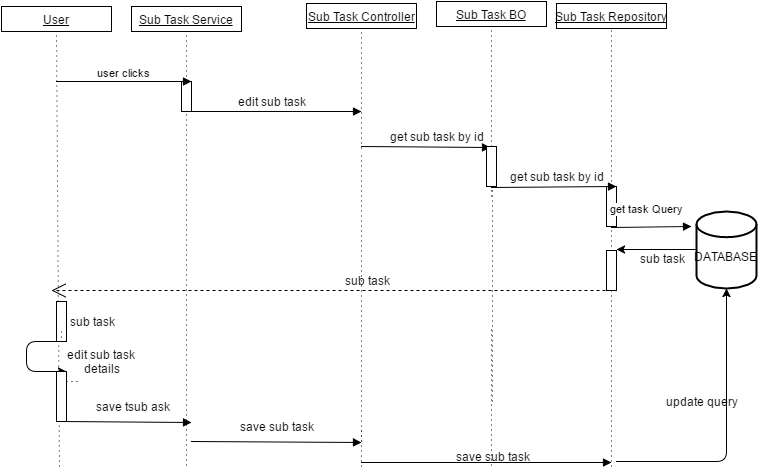
4.6.7 Add New Sub Task



4.6.8 Delete Sub Task

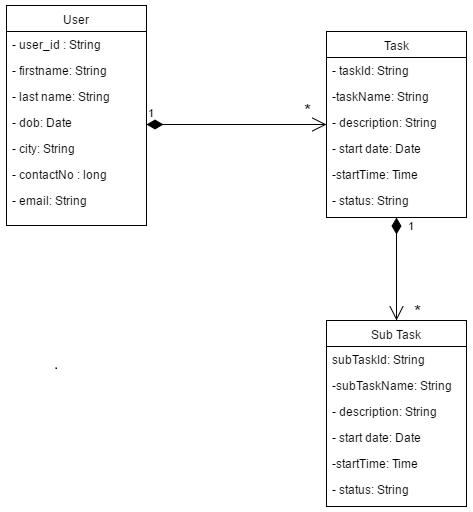


4.6.9 Edit Sub Task



**4.7** **Object** **Model**

Object Model is the description of the structure of the objects in a system including their identity, relationships to other objects, attributes, and operations.



|  |  |
| --- | --- |
| 5 Database Architecture |  |
|  |

The Task Management System Application will use MySql Database as its repository.

**5.1** **Data** **Model**

Data Model is a method for describing data structures and a set of operations used to manipulate and validate that data. Data Model for the Task Management System Application is as shown below –

**CREATE** **TABLE** `all\_user\_information` (

`user\_id` **VARCHAR**(50) **NOT** **NULL**,

`firstName` **VARCHAR**(20) **NOT** **NULL**,

`lastName` **VARCHAR**(20) **NULL** **DEFAULT** **NULL**,

`city` **VARCHAR**(20) **NULL** **DEFAULT** **NULL**,

`pin` **VARCHAR**(20) **NULL** **DEFAULT** **NULL**,

`dob` **DATE** **NULL** **DEFAULT** **NULL**,

`password` **VARCHAR**(50) **NOT** **NULL**,

`email` **VARCHAR**(50) **NULL** **DEFAULT** **NULL**,

`contactNo` **VARCHAR**(10) **NOT** **NULL**,

`profilePicture` **VARCHAR**(50) **NULL** **DEFAULT** **NULL**,

**PRIMARY** **KEY** (`user\_id`)

);

**CREATE** **TABLE** `login\_credentials` (

`srno` **INT**(11) **NOT** **NULL** **AUTO\_INCREMENT**,

`userId` **VARCHAR**(50) **NULL** **DEFAULT** **NULL**,

`password` **VARCHAR**(50) **NULL** **DEFAULT** **NULL**,

**PRIMARY** **KEY** (`srno`),

**INDEX** `login\_credentials\_ibfk\_1` (`userId`),

**CONSTRAINT** `login\_credentials\_ibfk\_1` **FOREIGN** **KEY** (`userId`) **REFERENCES** `all\_user\_information` (`user\_id`) **ON** **DELETE** **CASCADE**

);

**CREATE** **TABLE** `all\_tasks` (

`taskId` **INT**(11) **NOT** **NULL**,

`taskName` **VARCHAR**(20) **NULL** **DEFAULT** **NULL**,

`description` **VARCHAR**(20) **NULL** **DEFAULT** **NULL**,

`startDateTime` **DATETIME** **NULL** **DEFAULT** **NULL**,

`status` **VARCHAR**(20) **NULL** **DEFAULT** **NULL**,

`userId` **VARCHAR**(50) **NULL** **DEFAULT** **NULL**,

**PRIMARY** **KEY** (`taskId`),

**INDEX** `all\_tasks\_ibfk\_1` (`userId`),

**CONSTRAINT** `all\_tasks\_ibfk\_1` **FOREIGN** **KEY** (`userId`) **REFERENCES** `all\_user\_information` (`user\_id`) **ON** **DELETE** **CASCADE**

);,

**CREATE** **TABLE** `all\_sub\_tasks` (

`subTaskId` **INT**(11) **NOT** **NULL**,

`subTaskName` **VARCHAR**(20) **NOT** **NULL**,

`description` **VARCHAR**(20) **NULL** **DEFAULT** **NULL**,

`startDateTime` **DATETIME** **NULL** **DEFAULT** **NULL**,

`status` **VARCHAR**(20) **NULL** **DEFAULT** **NULL**,

`taskId` **INT**(11) **NULL** **DEFAULT** **NULL**,

**PRIMARY** **KEY** (`subTaskId`),

**INDEX** `all\_sub\_tasks\_ibfk\_1` (`taskId`),

**CONSTRAINT** `all\_sub\_tasks\_ibfk\_1` **FOREIGN** **KEY** (`taskId`) **REFERENCES** `all\_tasks` (`taskId`) **ON** **DELETE** **CASCADE**

);