**MyTasks.com**

**Manage Your Tasks and Project**

**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 MyTasks.com 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 MyTasks.com application’s 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 to demo use of AngularJs, Java script design patterns, HTML5 integration in a project implementation. Functionality is kept to bare minimum to avoid complexity. New flow can be added as and when needed for demo purpose.

**1.3** **Audience**

The intended audiences for this document are the faculty members and students.

|  |  |
| --- | --- |
| 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 understand and implement the best angularJs practices to for designing and developing a scalable web application.

This can be used to learn how to design and code a simple web application using java angularJs, HTML5 and CSS.

**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 and Practices used**

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 –

* **Revealing Module Pattern**
* **Repository Design Pattern**
* **Data Access Services**
* **$rootScope and $scope Services**
* **Custom Filters and Directives**

**2.4.1** **Revealing Module Pattern**

Revealing Module pattern encapsulates the JavaScript classes (functions) and reduces the spaghetti code. This pattern sometimes looks like an interface. This pattern ensures that all the methods and variables are private by default until they are explicitly exposed as public.  This pattern provides a very good structure to the JavaScript code and makes it is easier to understand and use. The custom services defined in the project use the module revealing patter, Functions that should be exposed publicly are defined in the return section of the angular.module.factory.

**2.4.2** **Repository Design Pattern**

Repository design pattern can be used in order to encapsulate into a separate layer all the underlying communication with a back-end remote system for performing CRUD operations. angularJs  is really a great open-source framework for creating single page web applications (SPA) based on MVW (Model / View / Whatever) architecture. It helps to implement a centralized entry point that controls and manages user (screen) request handling. angular.module.config handles the routing of view pages and loading the controllers according to it.

**2.4.3 Business Delegate pattern**

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. Angular..module.config act as business delegates in angularJS framework.

**2.4.4** **Data** **Access** **Services**

The application uses local storage to store the application data. AngularJs provides $localStorage service defined in ngStorage module of angular to store and retrieve data locally within the user's browser. Before HTML5, application data had to be stored in cookies, included in every server request. Local storage is more secure, and large amounts of data can be stored locally, without affecting website performance.

**2.4.5** **$rootScope and $scope Services**

Every application has a single root scope. The $rootScope is the most top-scope. It will be shared among all the components of application. Hence it acts like a global variable.

A $scope is JavaScript object which is used for communication between controller and view. Basically, $scope binds view to viewmodel and functions defined in a controller.

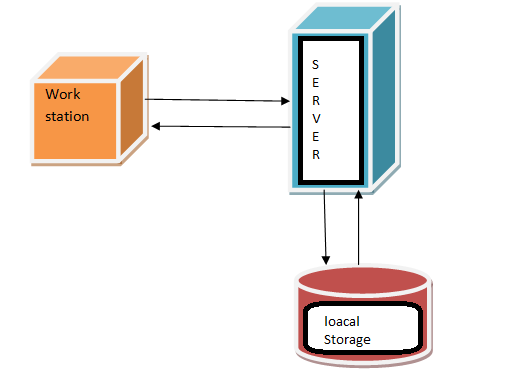
* + 1. **Custom Filters and Directives**

A filter formats the value of an expression for display to the user. They can be used in view templates, controllers or services and it is easy to define your own filter. You can make your own filters by register a new filter factory function with your module:

In addition to all the built-in AngularJS directives, you can create your own directives. New directives are created by using the .directive function.To invoke the new directive, make an HTML element with the same tag name as the new directive.

|  |  |
| --- | --- |
| 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 Banking Application –



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

**Web/ Application** **Server** – Tomcat 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.

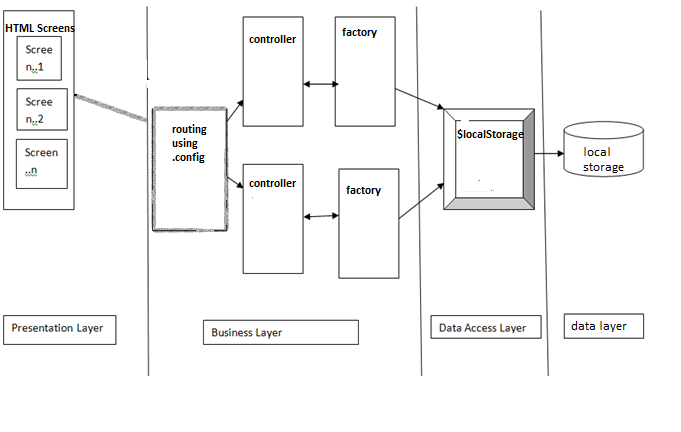
The server hosts the business logic and the business model classes of application as well. It serves requests for dynamic HTTP web pages from client.

**Local storage** - The application uses local storage to store the application data. AngularJs provides $localStorage service defined in ngStorage module of angular to store and retrieve data locally within the user's browser. Before HTML5, application data had to be stored in cookies, included in every server request. Local storage is more secure, and large amounts of data can be stored locally, without affecting website performance.

|  |  |
| --- | --- |
| 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:



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 Simple Banking Application. Presentation or view layer comprised of HTML 5 and CSS. Controller on user request invokes appropriate html page based on angular.module.config().

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

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

4.2.1 Action Classes: Action class 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. Transaction demarcation will be added on the services layer. 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 hibernate entity like Account, Person etc. 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.

Spring integration is used for wiring together these components.

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

Each Repository has session factory injected which is used to obtain a session which is used for data access from DB. Session factory uses schema provided by hibernate entities (having hibernate annotations) for Object relational mapping. Hibernate uses JDBC API underneath.

**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**

