Customer Bank Account Management System

Technical Specification Document
Table of Contents

Contents

1 Introduction.......................................................................................................................................................3
2 Design Overview.................................................................................................................................................4
3 Topology Diagram.............................................................................................................................................6
4 Application Architecture....................................................................................................................................7
5 Database Architecture.......................................................................................................................................14
1 Introduction

1.1 Purpose

The purpose of this document is to outline the technical design of the Banking 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 Banking 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 Struts2 (MVC pattern), Design patterns, Hibernate, Spring 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 through 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 J2EE application.

This can be used to learn how to design and code a simple web application using java technology stack, Struts2 framework, Spring integration, Hibernate 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 Value
- Object
- Command
- 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. FilterDispatcher acts as a front controller in Struts2.

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. Action classes act as business delegates in Struts2 framework.

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. Hibernate’s integration with spring forms data access layer of this application.

2.4.4 Value Object

The Value Object design pattern, also known as the Data Transfer Object, efficiently transfers remote, fine-grained data by sending a coarse-grained view of the data. This design pattern will be used for the communication between the middle tier and the back end. Hibernate entities when in transient/detached state will be used as DTOs in this application.
3 Topology Diagram

The diagram below provides an 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 their 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.

**MySql Database** – DB stores user profile, credentials and account 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.

JDBC – Java Database Connectivity is an application program interface (API) specification for connecting programs written in Java to the data in popular databases. Since we are using Hibernate we will not be using this API directly and don’t have to write SQL queries or code for object relational mapping explicitly.

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 JSP (HTML + standard tags) and a struts Controller. Controller on user request invokes appropriate JSP based on struts configuration. The JSPs also include JavaScript functions where applicable for input validation. 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 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 Repository(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
4.6 Sequence Diagram

In this section we detail the significant interactions between the major components for the Banking 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.

Note: The Sequence Diagram below is only a logical representation of the significant interactions of the system and may not directly map to the physical interactions of the system.
4.6.1 User Login

4.6.2 Forgot Password

4.6.3 Change Password
4.6.4 Register Payee

4.6.5 Transfer Funds

4.6.6 Manage User Account
4.6.7 Add New Account

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 Simple Banking 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 Simple Banking Application is as shown below –

```sql
CREATE TABLE `account` (
  `accId` int(11) NOT NULL auto_increment,
  `sum` float NOT NULL,
  `owner_id` int(11) default NULL,
  PRIMARY KEY (`accId`),
  KEY `FK1D0C220DBB52EA6F` (`owner_id`),
  CONSTRAINT `FK1D0C220DBB52EA6F` FOREIGN KEY (`owner_id`) REFERENCES `customer` (`id`)
);

CREATE TABLE `contact` (
  `id` int(11) NOT NULL auto_increment,
  `address` varchar(255) default NULL,
  `city` varchar(255) default NULL,
  `email` varchar(255) default NULL,
  `mobile` varchar(255) default NULL,
  `pin` varchar(255) default NULL,
  PRIMARY KEY (`id`)
);

CREATE TABLE `customer` (
  `id` int(11) NOT NULL,
  `dob` datetime default NULL,
  `gender` varchar(255) default NULL,
  `married` varchar(255) default NULL,
  `name` varchar(255) default NULL,
  `password` varchar(255) default NULL,
  `status` varchar(255) default NULL,
  `contact_id` int(11) default NULL,
  `occupation` varchar(255) default NULL,
  PRIMARY KEY (`id`),
  KEY `FK285FEB4AFD7C1027fbe3fe` (`contact_id`),
  CONSTRAINT `FK285FEB4AFD7C1027fbe3fe` FOREIGN KEY (`contact_id`) REFERENCES `contact` (`id`)
);

CREATE TABLE `employee` (
  `id` int(11) NOT NULL,
  `dob` datetime default NULL,
  `gender` varchar(255) default NULL,
  `married` varchar(255) default NULL,
  `name` varchar(255) default NULL,
  `password` varchar(255) default NULL,
  `status` varchar(255) default NULL,
  `contact_id` int(11) default NULL,
  `qualification` varchar(255) default NULL,
  PRIMARY KEY (`id`),
  KEY `FK285FEB4AFD7C1027fbe3fe` (`contact_id`),
  CONSTRAINT `FK285FEB4AFD7C1027fbe3fe` FOREIGN KEY (`contact_id`) REFERENCES `contact` (`id`)
);
```

PRIMARY KEY (`id`),
KEY `FK285FEB4AFD7C104afd4ace` (`contact_id`),
CONSTRAINT `FK285FEB4AFD7C104afd4ace` FOREIGN KEY (`contact_id`) REFERENCES `contact` (`id`);

CREATE TABLE `hibernate_sequences` (  
`sequence_name` varchar(255) default NULL,  
`sequence_next_hi_value` int(11) default NULL
);

CREATE TABLE `payee` (  
`id` int(11) NOT NULL auto_increment,  
`accId` int(11) NOT NULL,  
`payer_id` int(11) default NULL,  
PRIMARY KEY (`id`),  
KEY `FK4954328AA304E8D` (`payer_id`),  
CONSTRAINT `FK4954328AA304E8D` FOREIGN KEY (`payer_id`) REFERENCES `customer` (`id`)
);