**InfoABC Inc. – Expenses Share App**

**High Level Design (HLD) Document**

**Version 1.0**

**Prepared By**

**Niraj Tripathi**

# Revision History

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Version | Date | Change Description | Author | Reviewer |
| V1.0 | 27-May-2019 | Initial Draft Version | Niraj T. |  |
| V1.1 | 29-May-2019 | Minor changes | Niraj T |  |

Contents

[Revision History 2](#_Toc10012025)

[1. Introduction 4](#_Toc10012026)

[2. Design Specification 5](#_Toc10012027)

[2.1. Guiding Principles 5](#_Toc10012028)

[2.2. Project Structure 8](#_Toc10012029)

[2.3. Project Build & Deployment 9](#_Toc10012030)

# Introduction

InfoABC Inc., is a leading service provider operates worldwide with more than 100 vacations covering more than 70 countries in six continents. InfoABC has embarked upon a program to launch the Mobile App and APIs to provide widely-use consumer friendly web-services like Cost-Sharing App, which can solve the following pain-points for the users:

1. A single person tends to pay a bill (at a restaurant, etc) when they go out in a group and they expect to settle the sharing later.
2. When a group of people plan for a trip or similar multiple people spend on multiple items the cost of which needs to be shared later.

In such scenarios tracking of settlements becomes a challenge. What is needed is at any point in time a User should be able to know

* 1. What the user owes to others or vice versa
  2. Should be able to drill down and understand why the settlement amount is X.
  3. Should be able to view expenses for an event

1. Any further enhancements/extensions which can be thought of is a Plus. For these enhancements just presenting a design artefact should be enough.
2. Users can connect from Desktop, Tab, Mobile.

* Max people sharing in a group can be 1000
* Group can exist for more than a year, and can have partial settlements.

1. Expect concurrent 10000 requests for Expense addition + 5000 request for settlement + 2000 group creation requests.

This document describes high-level design specification for the Cost-Sharing App Project.

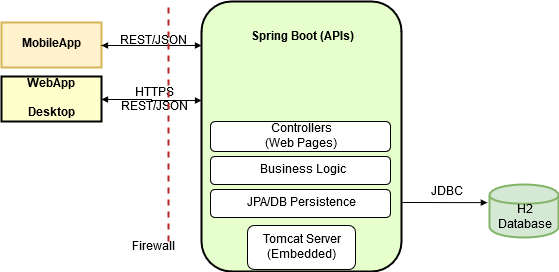


Figure 1: High-level Logical Architecture Diagram

The purpose of each layer for above diagram is described below:

**User Access Layer** – provides the user interface for end users, whether they are using desktop/laptop computers or mobility devices such as smart phones/tablet.

**DMZ Layer** – contains the system which reside inside the DMZ zone

**Web Application Layer** – contains the architecture’s application system reside inside the secured-network zone and protected by the security like Firewall port rules etc.

# Design Specification

As part of the project, following are the key interface required to meet the functional and non-functional requirements in-order to achieve the overall business need.

Given below table describes system information which are part of the project build & deployment implementation.

| Application Information | Remarks |
| --- | --- |
| H2 DB or  MySQL Server | Embedded Instance or Docker Image |
| Spring Boot | 2.1.5.RELEASE |
| Java/JDK8.0 | Java/JDK, JPA, Hibernate |
| 3rd Party Open source libraries |
| Web Server | Tomcat Server |

# Guiding Principles

Guiding principles provide a foundation upon which to develop the target architecture for the Application. Some of the guiding principles that will be followed during the design and development are outlined below.

1. **Scalable**

Scalability is the ability of the platform to scale both up and down to support varying numbers of users or transaction volumes. The application should be able to scale horizontally (by adding more servers) or vertically (by increasing hardware capacity or software efficiency).

1. **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.

1. **Standards-Based**

Web-services will comply with established industry standards. The standards-compliance will not only apply to application development but also to design, platform/infrastructure and other parts of the Integration. Examples of standards include REST, JSON, SOA and J2EE,

1. **Data Format**

JSON is considered as the default standard format for data exchanges with the service. However, support for XML format also would be provided by the service if there is a need for the same.

1. **URL Endpoint Design**

The API would be published using Swagger UI as REST endpoint for better maintainability covering multiple resources like User, Member, Expense, Category, MemberExpense etc. The URLs should be short and descriptive and utilize the natural hierarchy of the path structure.

Here are few examples:

* /users - Refers to the collection of Users
* /users/1234 - Refers to the particular User with Id 1234
* /users/1234/expense - Refers to the Expenses against the User with Id 1234

HTTP verbs are used to indicate operations on the resource. Here is the list of verbs and the operations they indicate:

| **HTTP Verb** | **Description** |
| --- | --- |
| GET | Retrieve resource |
| POST | Create/Insert new instance of a resource |
| PUT | Update existing instance of a resource |
| DELETE | To delete a resource instance |

The JPA based data persistence & repository service will be developed & available for enabling a dynamic mapping for DB Entity. The hibernate libraries used for the mapping and second-level cache usage.

1. **Error Handling**

The errors encountered by the service implementation could be categorized mainly into the following:

* Request related issue
* Error occurred during underlying processing

There are multiple mechanisms in the response that signal errors. First of all, the HTTP status would send a 2xx series value in case of success, a 5xx value if it is a server side error and a 4xx series value in case it is something that is wrong with the request and could be rectified by the client and retried. There is also a statusMessage which is the verbose version of the statusCode.

Here are some best practices to follow when returning an error response:

1. Return an HTTP status code that closely matches the error condition
2. Return human-readable error messages
3. Return machine-readable error codes
4. Return the Invalid parameter name

* 200 – OK (indicates the response contains a valid result)
* 400 - Bad Request or Invalid Input (indicates that the provided request was invalid. Common causes of this status include an invalid parameter or parameter value)
* 401 – Unauthorized (indicates that the service denied use of the API for wrong credentials)
* 404 - Not Found (indicates request data not found)
* 500 - Internal Server Error (indicates request could not be processed due to a server error. The request may succeed if you try again)

1. **Security**

Spring Security is used to provide web-page access using role-based permissions. The backend DB stores Users along with Roles for the API access.

The APIs will be protect using Basic authentication functionality with list of users defined under the database or external Identify Manager system.

# Project Structure

Given below diagram describes Spring Boot Project structure by listing the folder and files:

├── pom.xml

├── src

│   ├── main

│   │   ├── java

│   │   │   └── com

│   │   │       └── sharing

│   │   │           └── api

│   │   │            └── model

│   │   │            └── utils

│   │   │           └── etracker

│   │   │            └── controller

│   │   │            └── exceptions

│   │   │            └── model

│   │   │            └── repository

│   │   │            └── security

│   │   │            └── service

│   │   │            └── swagger

│   │   │            └── util

│   │   │           └── main

│   │   └── resources

│   │       ├── application.properties

│   │       ├── data.sql

│   │       ├── schema.sql

│   │       ├── validation.properties

│   │   └── webapp

│   │       ├── resources

│   │       ├── JSPs

│   └── test

│       ├── java

│       └── resources

└── target

# Project Build & Deployment

| **Feature/**  **Technology Stack** | **Description** |
| --- | --- |
| Web App | Java JSP, Spring Boot MVC with Spring-security |
| APIs | REST/JSON APIs |
| DB JPA | Backend DB processing using JPA with Hibernate |
| Others | POI Open source library to export the Report.  Also, there is a feature to upload the image to the App. |

Get Latest project from the github.com by using below GIT Repo URL:

<https://github.com/nirajtrip/cost-sharing-api.git>

Use Maven to build the project by running below command:

Mvn clean compile package

Use below command to launch the Application:

Mvn spring-boot:run

Open the Brower to view the web-app:

<http://localhost:8080>

Now, user can sign-up and login to the Web-app and create aGroup in order to store the Expenses against a group (e.g. Pizza Party). Backend processing logic will allocate amount to Group Members to determine amount Owned/Spent by the Member.

**Launching Swagger UI**

The Swagger UI available at

<http://localhost:8080/swagger-ui.html>

Also, The API Doc details available at

<http://localhost:8080/v2/api-docs>

To view the H2 Database Console, visit below URL:

<http://localhost:8080/h2-console>