Payment Gateway Solution

PG Architecture

tarang_new_logo_small.png

Tarang Software Technologies Private Limited

*Tarang Towers, #400/2, Whitefield Road, Hoody, Bangalore – 560048, India*

*Phone: +91-80-41157777; Fax: +91-80-41157870*

[www.tarangtech.com](file:///D:\Users\Vijay\AppData\Local\Microsoft\Windows\Temporary%20Internet%20Files\Content.Outlook\9KP06ZQY\www.tarangtech.com)

Contents

[1 Introduction 5](#_Toc330466601)

[2 Key Architectural Considerations 5](#_Toc330466602)

[2.1 High performance 5](#_Toc330466603)

[2.2 Scalable 5](#_Toc330466604)

[2.3 Extensibility 5](#_Toc330466605)

[2.4 Loose Coupling 5](#_Toc330466606)

[2.5 High Availability 5](#_Toc330466607)

[2.6 Security 6](#_Toc330466608)

[2.7 Audit Trail 6](#_Toc330466609)

[2.8 Usability 6](#_Toc330466610)

[2.9 Maintenance 6](#_Toc330466611)

[3 Approaches to Architecture 6](#_Toc330466612)

[3.1 Merits and Demerits of Architecture Approach 7](#_Toc330466613)

[4 Logical Architecture 9](#_Toc330466614)

[4.1 Client Tier 9](#_Toc330466615)

[4.2 Web Tier 10](#_Toc330466616)

[4.3 Application/Business Tier 10](#_Toc330466617)

[4.4 Data Tier 10](#_Toc330466618)

[4.5 OLTP TXN Listener 10](#_Toc330466619)

[5 Architectural Layers 11](#_Toc330466620)

[5.1 Client layer 12](#_Toc330466621)

[5.2 Application/Business layer 12](#_Toc330466622)

[5.3 Common layer 12](#_Toc330466623)

[5.4 Integration layer 12](#_Toc330466624)

[6 Physical Architecture 12](#_Toc330466625)

[6.1 APP Servers 13](#_Toc330466626)

[6.2 Oracle APP Servers 13](#_Toc330466627)

[6.3 Storage 14](#_Toc330466628)

[6.4 L3 Switch 14](#_Toc330466629)

[6.5 Leased Line 14](#_Toc330466630)

[7 Entity Relationship Diagram 15](#_Toc330466631)

[8 Software Environment 15](#_Toc330466632)

[8.1 Software Stack 15](#_Toc330466633)

[9 Non functional Requirements 16](#_Toc330466634)

[9.1 Performance Requirements 16](#_Toc330466635)

[9.1.1 Scalability 16](#_Toc330466636)

[9.1.2 Redundant and Load Balancing 16](#_Toc330466637)

[9.1.3 Partitioning by function 17](#_Toc330466638)

[9.1.4 No sticky sessions 17](#_Toc330466639)

[9.1.5 Limiting database transactions 17](#_Toc330466640)

[9.1.6 Moving processing to asynchronous flows 17](#_Toc330466641)

[9.2 Availability 17](#_Toc330466642)

[9.3 Failover 17](#_Toc330466643)

[9.4 Auditing 18](#_Toc330466644)

[9.5 Security Requirements 18](#_Toc330466645)

[9.5.1 Network Threats and Countermeasures 18](#_Toc330466646)

[9.5.2 Host Threats and Countermeasures 18](#_Toc330466647)

[9.5.3 Application-level Threats and Countermeasures 19](#_Toc330466648)

[9.5.4 Usability 21](#_Toc330466649)

**DOCUMENT CHANGE HISTORY**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Author** | **Date** | **Reviewed By** | **Date** | **Version** | **Description** |
| Tarang | 18-07-12 |  |  | 1.0 |  |
|  |  |  |  |  |  |

# Introduction

Online shopping allows acceptance of credit card payments in a card-not-present environment. This is made possible by a payment gateway. Payment gateway is an e-commerce application that provides transaction authorization and clearing services to Internet based payment portals such as a website, mobile phone or IVR service. The gateway encrypts transaction information and transmits it between the website and the merchant’s acquiring bank. This document describes the architecture of the Tarang Payment Gateway solution.

# Key Architectural Considerations

The technical architecture is driven through the following major considerations:

## High performance

The system should have high throughput and the response time should not vary even under higher loads. The targeted performance is measured in terms of the time taken for processing a typical TXN to be between 2 to 3 seconds. The TXN engine is critical component which along with other helper components, will serve the customers.

## Scalable

The system should be able to scale very easily to meet higher loads allowing plug ‘n’ play of hardware and software components. The system should be able to scale up either vertically and / horizontally and should perform well.

## Extensibility

This is a desired feature for product development where in if needed, a service / product can be bought from a third party source and/or integrated. The architecture should allow this, making the product more modular.

## Loose Coupling

The system components are loosely coupled with each other and interact using definite interfaces each exposes. The TXN processing engine is a separate module using configuration and settings provided using various consoles. The modules interact through DB, thus driving loose coupling yet giving optimal performance. Also loose coupling will be implemented at layering the architecture in multiple tiers and segregating presentation, web, application and database layer. All system components at physical architecture level also will interact in a loosely coupled manner.

## High Availability

The system is expected to be available all the time (99.99%). The proposed deployments foresee this and we have the necessary clusters for key components like TXN engine, settlement & recon and web consoles.

## Security

The system is accessed by Merchants and Customers for TXN processing. These entities access the system over HTTPS. The security module within the system will provide with necessary API for encryption / decryption.

## Audit Trail

The system keeps an audit trail of all the financial TXN processed. Also the changes made to configuration, business rules, secure info about any stakeholder in the system, are put in audit. The system also presents these trails as reports to the privileged users of the system.

## Usability

1. Merchant processes financial transaction for customer who is willing to pay for their services/products
2. Multiple communication channels are supported for merchants to process transactions
3. Different payment models shall be made available for merchants
4. Merchants have easy to use, intuitive Web console to interact with system as well as manage respective business.
5. Internal employees are provided with easy to use, intuitive Web console to interact with system as well as manage respective business.

## Maintenance

The Customer must be able to operate with minimum interruptions and complexity. The system provided various Ops consoles so that the system can be governed easily and providing audit trails, to look into problems.

# Approaches to Architecture

The system can be architected in the following ways:

1. **Set of applications** In this approach, the set of functionalities are designed as applications. The applications are integrated through DB. Though this approach offers flexibility, there will not redundant code, required to do common functionalities in every app. Also the system cannot scale up/down easily, since, the apps work individually. Also configuration and business rules validation will be very difficult as multiple apps (silos) can take part in the process flows spawning across the system. High performance will be a problem since each app works in its own boundary.
2. **Group of components** As a component based system with module for each domain model. These components can be housed in two silos, to drive TXN processing and Web parts. The components can have common modules to help with utility or plumbing functionalities. But in this way, we will redundant plumbing code in modules. Also we cannot plug ‘n’ play modules, since they are not working as services, exposing definite API. We cannot easily extend or scale up / down.
3. **SOA** In this approach the complete system works on messages passed on through web services. The performance will badly hit, since marshaling and un-marshaling takes lot of time. Also it will have lot of plumbing code in every module/service to receive request and send response. This approach is suitable only when there are multiple external systems are involved while processing.
4. **Modular Services** This approach considers each module as a service, concentrating on particular set of business functionalities. The modules have clear API to interact with each other. One or more modules together form a component. The components need not housed together and integrated through DB. The services are configurable through parameters and business rules can be applied on service or set of services. The common functionalities like alerts work asynchronously. This approach offers desired flexibility, extensibility and can easily scale up / down. This approach also offers the desired loose coupling. Now that components can be scaled up as needed and code is optimized per component level, high performance can be assured.

The system is architected as a highly scalable, high performance, easily extendable platform. We are adopting the choice #4.

## Merits and Demerits of Architecture Approach

The following are the merits of approach:

1. Since the modules are made of services, the system will be highly configurable
2. Since each module offers definite API to interact with other modules, the system will be more flexible. Also modules can integrated into a component, thus giving more flexibility at a minimum cost
3. Modules can integrated into a component, the system can be scaled up / down easily
4. The common plumbing code can grouped into a module or set of services and can be housed in a separate server, thus allowing for easy maintenance and flexibility
5. Where ever possible, the service configuration and business rules are parameterized and stored in DB. So the integration is done through DB
6. Since system is modularized and service driven, allows us to have service level code optimization

The following are demerits:

1. It takes more time and resources to architect and design the system.
2. It requires more in depth knowledge and skill for integrating services into modules and components.

**Mitigation** of perceived demerits in the architecture

1. We have pooled in highly experienced resources for architecture and design
2. We are putting in more time to make robust architecture with complete platform in perspective. Proof of concepts shall be done to test the waters and correct the system accordingly to meet the perceived goals.
3. The architecture, high level design will address more details and process flows, there by complexity is reduced while implementing and testing the platform

# Logical Architecture

Given below is the logical architecture diagram which describes how different tiers and services interact with one another. It also serves to show different components identified as a part of the product suite.

The above depicts the technical view into the tiers of the architecture. It has the following tier components:

1. Client tier
2. Web Requests listener
3. Mobile APP / IVR / WAP listener
4. Business tier
5. Data tier

## Client Tier

The client layer includes interactions from browsers, devices (ATM, Mobile-Telco transactions and transaction terminals), and any 3rd party applications which are supported by the system. Financial institutions switches or POS/ATM transaction acquirer shall redirect all the transactions to this system based on bin ranges in predefined formats.

## Web Tier

Each of the end interfaces would have a presentation layer which would define user interactions using JSP/jQuery/AJAX/etc technologies. The presentation layer would interact with business layer/services in a secure manner using SSL.

## Application/Business Tier

Different set of business processes have been identified to be exposed as a APIs by each identified components. Each component would expose APIs for other components to request/update information.

## Data Tier

The above-mentioned applications and services will use following database to store their data. Following details are stored in database.

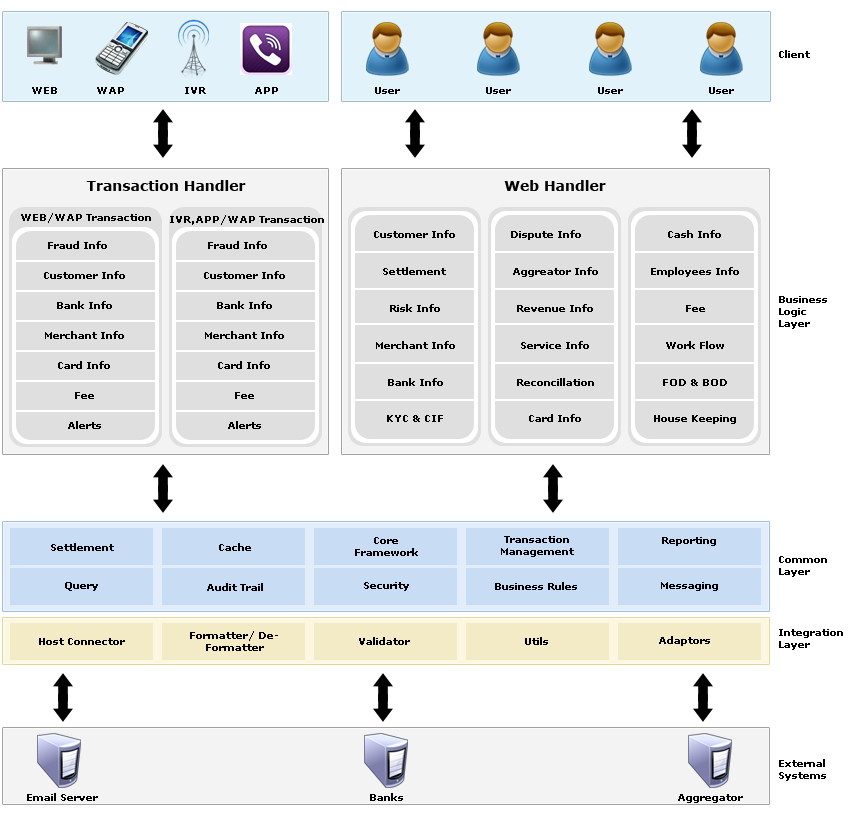
* All information related to the merchants and associated entities information is stored in this like merchant details, KYC confirmation flows, work flow etc.
* All information related to the bank and associated entities information is stored in this like Bank details, Recon Schedules, Charge back info etc.
* All information related to the user entities identified and supported by the system are stored in this db. This db stored information related to customers, their transactions, personal information which has been submitted by merchants during transaction processing etc.
* All information related to the financial processing in/through the system shall be stored
* All messages (confirmations, alerts, notifications and reminders), audit trails and transaction logs will be stored in this database which will enable message transfer across components, products and external systems.

## OLTP TXN Listener

Different set of business flow has been identified for this specific listener to meet the performance and reliability requirements.

# Architectural Layers

Given below are the Application Architecture Diagram that gives a perspective of technology stack and the architecture being used in a layered n-tier model.



The above diagram shows the various logical components within the system. The following are the details of each component:

1. Client layer
2. Business logic layer
3. Common layer
4. Integration layer
5. External system interacting with the system

Each of the products/services encompasses its own client-tier, web-tier, business-tier and data-tier layers. Each product will use exposed business services, apart from reusing some of the common services published in the business-tier and any common tables and entities in the data-tier.

These products can run in the same JVM or on different JVMs with each product collaborating with the other using the services exposed in the business-tier. The Services are nothing but an abstraction layer which handles the communication with the internal databases and external integration requirements with systems.

## Client layer

This layer consists of the client browser or Mobile phone Application or IVR. The users would interact with the system using these and would receive the response on the same. Each of these will be a ‘channel’ for communicating with the system.

## Application/Business layer

The requests from channels, WEB and High end WAP Phones would be served by the transaction handler component of the application layer. The transaction handler would receive the request and process it. While processing it uses various configuration and static info from the entities involved and stores the results persistently into the DB. The processing would keep the necessary audit trail of the process, for the operations support.

## Common layer

The common layer represents the common functionalities used in the system by various functional modules. For example the audit module would help all other modules keeping tracking of audit across the system. The settlement module is used by all channels transactions.

## Integration layer

The integration layer represents the integration with external systems and interfaces. For example, Sale transaction, the transaction details are sent to bank through bank interface, the response analyzed, processed and response can be sent back. In future if the customer personal details needs to verified through a third party based on PAN or UID, could also be easily done with minimal coding.

# Physical Architecture

Given below is the physical architecture which defines the various system components where application/database are installed or accessed from, and with recommended configuration, connectivity options will meet the standards in the context of security, reliability, and performance along with Scalability.

## APP Servers

* There will be three APP servers; two will be at Primary Site and one at DR Site
* Server configurations will be:
  + Dual Hexa Core (**Intel E5645**) Processor
  + 32GB RAM
  + 2 x 300GB 15K SAS HDD with Raid1+0
  + 4 x 1Gb Eth ports
* VMware or RHEV will be used on APP servers for virtualization. There will be 6 VM’s on each of the APP servers i.e.
  + **OLTP** – 7 GB RAM – RHEL/CentOS (preferred RHEL )
  + **Portal** -8 GB RAM – RHEL/CentOS (preferred RHEL)
  + **VM Hypervisor** – 1GB RAM – VMWARE or RHEL
  + **Apache Load Balancer** – 3 GB RAM – RHEL/CentOS
* Total 19 GB RAM will be used out of 32GB keeping 13 GB in buffer
* Connectivity to storage depends on budget (iscsi or SAN)

## Oracle APP Servers

* There will be three Oracle App servers; two will be at Primary Site and one at DR Site
* Server configurations will be:
  + Single Quad Core (**Intel E5620)** Processor
  + 32GB RAM
  + 2 x 146GB 15K SAS HDD with Raid1+0
  + 2 x 1Gb Eth ports
  + 1 x Dual port HBA card (FC interface) or 2 x Single port HBA card (FC interface)
* Oracle app servers with configured in HA active/standby mode on RHEL cluster at primary site.

## Storage

* There will be two storages; One will be at Primary Site and one at DR Site
* Storage configurations will be:
  + Dual Controller Array
  + FC Based
  + Storage will have 12 x 300GB 15k SAS HDD’s configured with Raid 1+0
  + Hot-Plug HDD (preferred)
* Storage-to-Storage replication will be used for the data replication between storages at different site.

## L3 Switch

* Three L3 Switched will be installed, Two at primary site and one at DR Site.
* Different VLAN will be created for multiple purpose Data transfer

## Leased Line

* There will be a Ethernet drop Leased Line connectivity between Primary and DR site. Leased line Bandwidth will be 2Mbps and is enough as per the required TPS.

# Entity Relationship Diagram



DB Design attached is high level and this might be enhanced to meet the requirements during the customization stage.

# Software Environment

## Software Stack

Following are the identified tools and technologies on which product will be built.

|  |  |  |
| --- | --- | --- |
| **Requirement** | **Tools/ Services** | **License** |
| **Application / Service Tier** | |  |
| Java | Java 1.7.X - "Java" generally refers to a combination of three things: the Java programming language ; the Java Virtual Machine (a high-performance virtual machine that executes bytecodes on a specific computing platform, typically abbreviated *JVM*); and the Java platform, a JVM running compiled Java bytecodes | Open Source |
| Web Server | [Apache](http://httpd.apache.org/) – This is a web server, which will be used to front application server and will server static content as well. | Open Source |
| Dependency Injection | [Spring](http://www.springframework.org/) – Spring is the most popular Dependency Injection framework widely used in enterprise application for loosely coupling applications / services together. | Open Source |
| Scheduler | [Quartz](http://www.opensymphony.com/quartz/) – This is a scheduler for Java somewhat similar to \*nix cron jobs.  Oracle Background Jobs. | Open Source |
| Spring Security | [Spring Security](http://www.acegisecurity.org/) - This is a open source framework to implement authentication and authorization. If required. | Open Source |
| **Backend Systems** | |  |
| Database | **Oracle 10g** – Oracle is the most advanced RDBMS system, used to store relational data. Stored procedure queries will be written using pl/sql for performance improvement where business logic is data intensive. | Commercial |
| **Client Tier** | |  |
| JQUERY | JQUERY framework for Java. | Open Source |
| JSP | JSP | Open Source |
| **Development Tools** | |  |
| Development Tool | [Eclipse](http://www.eclipse.org/) – The leading open source Integrated Development Environment for Java application development. | Open Source |
| Source Control | **Subversion** [SVN](http://subversion.tigris.org/) – This is used to manage the versions of source code. | Open Source |
| Defect Management | Bugzilla– This is a bug tracking, issue tracking tool | Open Source |
| Browsers | Windows - IE 7 and higher, Firefox latest, Google Chrome 12 | Open Source |

# Non functional Requirements

## Performance Requirements

The performance requirements of the application are mentioned below-

* Application uptime must be > 99.99%, including hardware/software upgrades and maintenance.
* All non-transactional pages must be rendered in less than 6 seconds.
* The application should not take more than 3 seconds for processing device requests.

To meet these performance requirements, concurrent programming APIs will be used wherever possible. The data, which is generally used by most of the users, will be pre-fetched in background and cached in memory (in session or shared cache) to give better perceived performance to the end users.

On the client side (browser) JQuery will used to minimize the entire page refreshes.

### Scalability

Due to the JVM’s limitation on the amount of memory acquisition, scale-out architecture is the preferred solution over scale-up architecture.

In order to scale out, an application will be created such that each element that runs on a server does not contain state information. This allows it to exploit added resources in a scalable manner. In other words, to scale out well, an application should be parallelized so that different parts are independent of other parts, with each part taking advantage of appropriate resources.

Scalability will typically require deploying multiple server instances in a clustered environment. Also application also will be geared for scalability by taking care of the following -

### Redundant and Load Balancing

Products and services will be deployed in redundant to have better performance and failover mechanism. All products and services are designed to be able to run in separate instances, which provide flexibility to scale an individual product / service according to the amount of traffic it attracts. To divide the load among servers equally or as per their strength the Load Balancer will be used to get optimal performance out of the servers. If traffic increases substantially then hardware based load balancers will be evaluated based on the problems we encounter in production.

### Partitioning by function

The system will be composed of loosely coupled components which cater to various functional requirements of the system. The services will be designed to be stateless and to have related pieces of functionality belonging together. This allows the system to scale each service pool independently of one another, according to the demands and resource consumption of its function.

At the database tier, the same approach can be taken to separate related data into different database instances. This approach allows scaling the database infrastructure for each type of data independently of the others.

### No sticky sessions

Using session replication, the presentation layer can be designed to avoid server affinity. No transaction state should be maintained on the application tier, facilitating the optimal distribution of load across the servers.

### Limiting database transactions

Database transactions should be used only when absolutely required to reduce the resource lockups. Distributed transactions using two-phase commit comes with substantial costs. Scaling, performance, and latency are adversely affected by the costs of coordination, which worsens geometrically as the number of dependent resources and incoming clients increase.

### Moving processing to asynchronous flows

Between components, synchronous coupling will be avoided as much as possible. Moving as much processing as possible to asynchronous mode will decrease the user latency. In a system where replying rapidly to a request is critical, this can substantially reduce the latency experienced by the requestor.

Asynchrony can substantially reduce infrastructure cost. Performing operations synchronously forces scaling the infrastructure for the peak load - it needs to handle the worst second of the worst day at that exact second. Moving expensive processing to asynchronous flows, though, allows scaling the infrastructure for the average load instead of the peak. Instead of needing to process all requests immediately, the queue spreads the processing over time, and thereby dampens the peaks.

## Availability

To achieve high availability all products will be deployed in multiple instances. So that if one node (server) fails due to any reason others will keep on serving the clients though with slightly degraded performance. The databases will also be replicated in real time so that backup database can replace the primary server with least downtime.

## Failover

All products will be deployed in small instances with session replication across the physical machines. Session stickiness will be used so that a user is served by the same cluster during his entire session. This session stickiness and replication will make failovers transparent to the user as if one node (server) fails then another node in the same cluster will be able to continue to keep on serving user requests without any interruptions.

## Auditing

All financial transactions will be logged at a separate place other than their designated database tables, which can be used when some disputes arise. Database triggers will create essential transaction logs at each occurrence of transaction. Audit logging will be done asynchronously using effective techniques with cache mechanism as base so that it does not impact the overall application performance.

## Security Requirements

Since the system handles sensitive information like customer personal information, balance information, and interacts with payment gateways/bank networks, the security requirements are stringent to prevent any kind of data thefts and attacks.

The system has multiple levels of access privileges e.g. admin, member etc, so access control is of paramount importance in the application architecture.

The various potential vulnerabilities can be broadly classified into three categories:

* + Network infrastructure vulnerabilities
  + Host-level vulnerabilities
  + Application-level vulnerabilities

The above-mentioned categories are further elaborated below, with the various threats and countermeasures.

Network and Host level vulnerabilities will be overcome during the deployment of application in the production environment. However Application-level vulnerabilities are factored during design and development of the product.

### Network Threats and Countermeasures

The primary components that make up the network infrastructure are routers, firewalls, and switches.

An attacker may exploit poorly configured network devices using techniques like packet sniffing, IP spoofing, foot printing and denial of service attacks.

To prevent these attacks, countermeasures include:

* Configuration of the network devices to restrict foot printing requests.
* All software involved in the network level is to be updated with the latest security patches.
* All unused protocols and ports need to be blocked.
* Encryption techniques need to be used for secure communication channels.
* IDS tools need to be used to counter DoS attacks.

### Host Threats and Countermeasures

Host threats are directed at the system software upon which the applications are built. This includes the operating system software, web server, application server and database.

Host level threats like foot printing and unauthorized access can be mounted on the system by using port scans, ping sweeps, and accessing default accounts on the software.

These threats can be countered by taking measures such as:

* Unnecessary protocols and ports will be disabled.
* IDS can be configured to pick up foot printing patterns and reject suspicious traffic.
* Directory browsing option in web and application server will be disabled.
* Default accounts in the OS, web server, application server and database servers will be disabled.

### Application-level Threats and Countermeasures

#### Password Cracking

**Threat**

An attacker can establish an authenticated connection by guessing a valid username and password combination. The use of blank or weak passwords by users can provide an attacker easy access to the system.

**Countermeasures**

* Strong password policies will be used.
* Passwords will be stored in non-reversible password hashes in the database.
* Lockout policies will be used on accounts to limit the number of retry attempts that can be used to guess the password.
* Audit of failed logins for patterns of password hacking attempts will be maintained.
* Product will use this JCaptcha (Can be integrated with Spring Security) for user registration to prevent brute force password cracking attacks and also to detect usage of requests made by robots or other automated processes.
* All critical user information like Card number must be masked when displayed in browser
* All critical user information like user credentials, account number, expiry date etc. must be stored in encrypted/hashed form in database.

#### SQL Injection

**Threat**

A SQL injection attack exploits vulnerabilities in input validation to run arbitrary commands in the database. It can occur when the application uses input to construct dynamic SQL statements to access the database. It can also occur if the code uses stored procedures that are passed strings that contain unfiltered user input. The issue is magnified if the application uses an over-privileged account to connect to the database. In this instance it is possible to use the database server to run operating system commands and potentially compromise other servers, in addition to being able to retrieve, manipulate, and destroy data.

**Countermeasure**

* Thorough input validation will be performed prior to sending a request to the database.
* Usage parameterized stored procedures for database access will be enforced to ensure that input strings are not treated as executable statements. If stored procedures cannot be used, SQL parameters will be used when for building SQL commands. PreparedStatement objects in Java provide this facility.

#### Cookie Replay Attacks

**Threat**

With this type of attack, the attacker captures the user's authentication cookie using monitoring software and replays it to the application to gain access under a false identity.

**Countermeasures**

* Encrypted communication channel will be used provided by SSL whenever an authentication cookie is transmitted.
* We shall avoid usage of client side Cookie.

#### Query String Manipulation

**Threat**

Users can easily manipulate the query string values passed by HTTP GET from client to server because they are displayed in the browser's URL address bar. If the application relies on query string values to make security decisions, or if the values represent sensitive data such as monetary amounts, the application is vulnerable to attack.

**Countermeasures**

* Using query string parameters that contain sensitive data or data that can influence the security logic on the server will be avoided.
* HTTPS POST will be used instead of GET to submit forms.

#### Week Session Cookie Manipulation

**Threat**

Cookies are susceptible to modification by the client. A number of tools are available to help an attacker modify the contents of a memory-resident cookie. Cookie manipulation is the attack that refers to the modification of a cookie, usually to gain unauthorized access to a Web site.

**Countermeasures**

* Minimum client information will be stored in the cookies.
* We shall avoid storage of any data in the client-side cookies.
* Cookies will be encrypted with strong 256-bit encryption (subject to the information that is being saved) in unavoidable scenario’s.

#### Exception Management

**Threat**

Exceptions that are allowed to propagate to the client can reveal internal implementation details that make no sense to the end user but are useful to attackers. Applications that do not use exception handling or implement it poorly are also subject to denial of service attacks.

**Countermeasures**

* Exception handling will be used throughout the application's code base and logged.
* Generic error messages will be returned to the client, which will not reveal any internal implementation details.

#### Authentication and Authorization

Authentication and authorization will be implemented using Spring Security framework. Spring Security provides comprehensive security services for J2EE-based enterprise software applications.

At an authentication level, Spring Security supports a wide range of authentication models such as JDBC, JAAS, LDAP, HTTP Digest, CAS, JOSSO, automatic "remember-me" authentication, and has container integration capability with Tomcat. The system will be using the database-based authentication mechanism for authenticating users.

Product will use both the web-request interception and method-invocation interception techniques for handling authorization.

The Blowfish encryption algorithm will be used for secure data transfer wherever applicable. Blowfish is a fast, symmetric cipher with no patent restrictions, and has a small memory footprint.

For one-way hash functions, SHA-1 will be utilized.

Session timeout (no activity) will be 15 minutes.

### Usability

AJAX and Reverse AJAX along with DHTML, CSS will be used for page refreshes and pushing data from server. Scriptaculous will be used for adding behavior and effects on the web page. There will be predefined templates based on which user can change the look and feel of the application.