**Local Search**

**Technical Design Document (TDD)**

for the

**Local Search Engine Information System**

**Version: 0.96**

# Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author(s)** | **Summary of Changes** |
| 0.90 | 3-Apr-2007 | Svetlin Nakov | Initial draft |
| 0.91 | 10-Apr-2007 | Svetlin Nakov | Updated the architecture diagram |
| 0.92 | 16-Apr-2007 | Svetlin Nakov | Updated the architecture diagram;  Defined internal module design for the most important modules;  Defined technologies and alternatives for each module |
| 0.93 | 17-Apr-2007 | Svetlin Nakov | Minor changes after discussion with Angel Sotirov |
| 0.94 | 19-Apr-2007 | Svetlin Nakov | Added security to the modules design |
| 0.95 | 25-Apr-2007 | Svetlin Nakov | Added the database diagram for the Core DB |
| 0.96 | 17-April-2007 | Svetlin Nakov | Redesigned the concept of modules, facades and services |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# About This Document

The purpose of this document is to describe the architecture of the Local Search search engine, its modules, services and their interaction. This includes the global system architecture overview, internal design of the modules, the database design, communication protocols and the technologies stack.

# Referenced Documents

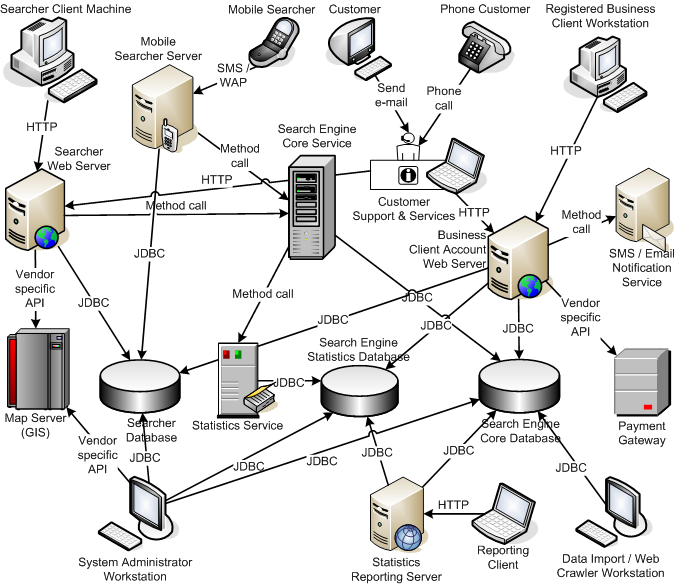
|  |  |  |  |
| --- | --- | --- | --- |
| **Document Name** | **Version** | **Date** | **Author(s)** |
| LocalSearch-Software-Requirements-Specification.doc | 0.9 | 2-Apr-2007 | Svetlin Nakov,  V. Tsanev |
| GeneralViewModel.pdf | - | - | A. Sotirov |
|  |  |  |  |

# Architecture Overview

The Local Search system uses multi-tier distributed software architecture with several front-end servers, back-end servers and databases. It consists of Web servers for the searchers, Web servers for the registered business clients, Core search engine module, accessing the Core Search engine database and a Search Statistics module. Additionally the system supports GIS server to visualize locations at the Web front-end and payment gateway module, data import modules and administrative modules running on separate machines.

## General Architecture Diagram

The following diagram shows the technical infrastructure of the system and interaction between different subsystems and different actors:



## Physical Infrastructure

The architecture diagram above shows possible physical infrastructure for the whole system. Note that several modules can run as services on the same physical server in the same JVM.

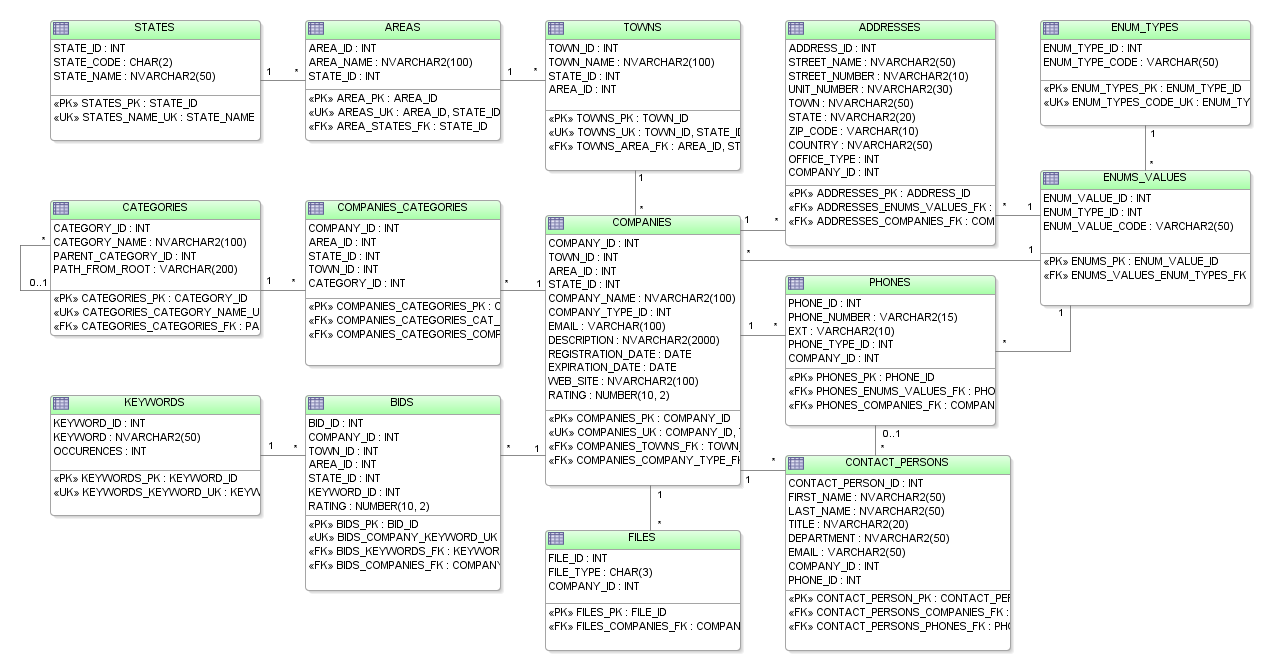
# Database Design

This section describes the databases that will be used by the modules in the system and provides E/R diagrams.

## Search Engine Core Database

The search engine core database contains the business entities (companies), keywords, categories and all other information that is searched by the searchers.

### E/R Diagram



### Entities Details and Constraints

#### Enumerations

Enumerations are sets of constant values used in different tables, e.g. company type which can be one of the following: Ltd., Corp., Inc., LLC, etc.

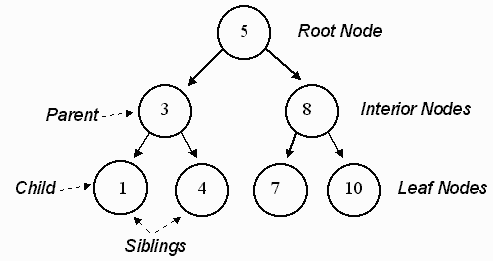
We store all enumerations in the database in the following tables: ENUM\_TYPES, ENUM\_VALUES.

ENUM\_TYPES table holds a single entry for each enumeration type, e.g. "company type", "phone type", "office type".

ENUM\_VALUES table holds a set of values for each enumeration type. The values are identified by the ENUM\_VALUE\_CODE column.

#### Categories

The categories table has some special constraints. The field PATH\_FROM\_ROOT holds the path from the root to the current node as sequence of CATEGORY\_ID numbers separated by arrow "->". For example consider the following tree:



The root node has PATH\_FROM\_ROOT="5". The node 7 has PATH\_FROM\_ROOT="5->8->7". The node 3 has PATH\_FROM\_ROOT="5->3".

This allows finding all subcategories of given category (e.g. node 3 from the above figure) recursively with single select like this:

|  |
| --- |
| **select \* from CATEGORIES where PATH\_FROM\_ROOT like "5->3->%"** |

#### States, Areas and Towns

In the database each town is located in some area and each area is located in some state. For example we can have the town York, located in Toronto area which is in the state of Ontario (ON). Some towns could have no area (e.g. Brampton, ON). We assign them to the global area of their state which is named by the pattern **<STATE>\_GLOBAL**, e.g. **Ontario\_GLOBAL**.

## Search Engine Statistics Database

...

### E/R Diagram

...

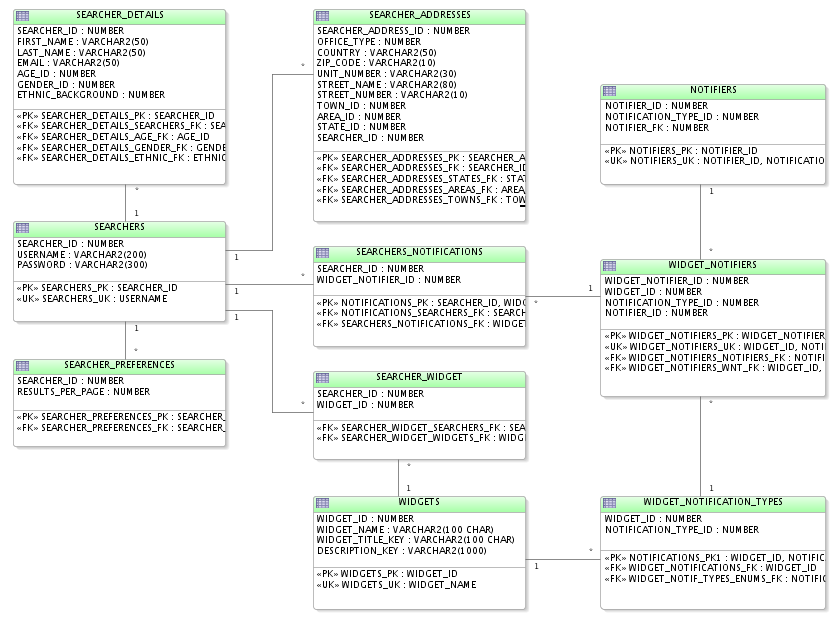
### Entities Details and Constraints

...

## Searchers Database

...

### E/R Diagram



### Entities Details and Constraints

...

# Modules Architecture and Internal Design

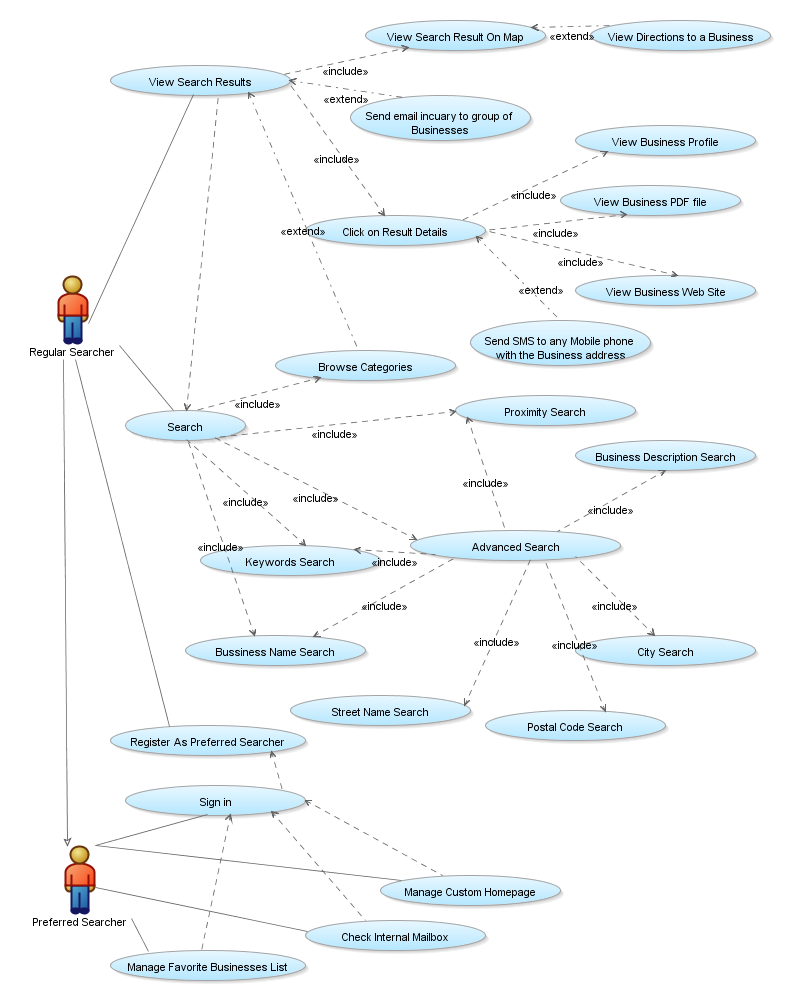
This section provides architecture and internal design description of the individual modules and services of the system. This includes architecture diagrams, technologies overview, and detailed diagrams that defines the structure of the modules and describes the processes.

## Module Searcher Web

The Searcher Web Front-End Module provides Web based user interface to the Internet users and allows them to perform searches in the search engine. Searchers are allowed to login in the Searcher Web Module and handle their profiles.

The Searcher Web Module is running inside a Web server and can host other modules of the system running as services inside its JVM.

### Use Cases



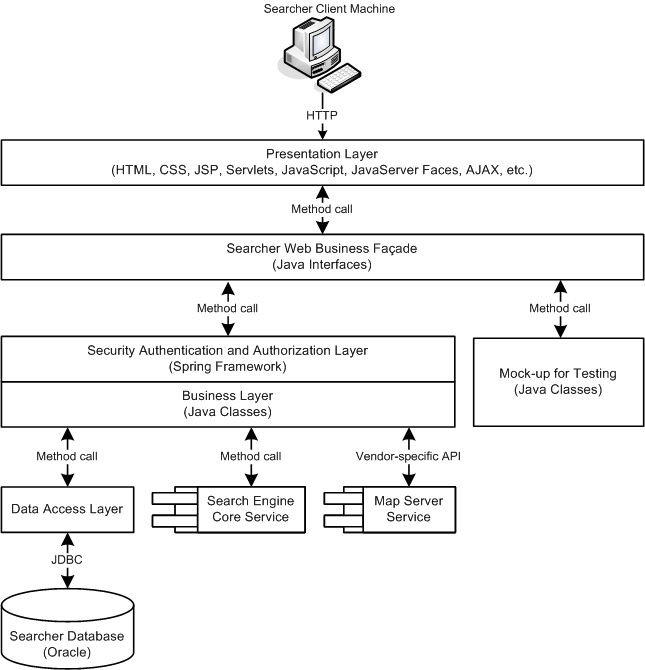
### Key Requirements and Design Considerations

The module design conforms to the following key requirements:

* Performance – should be able to handle thousands requests in the same time.
* UI Performance – Web interface reasonable performance.
* Small Bandwidth Consumption – not too much network load for the Web front-end.
* User-friendly Web based interface – should provide AJAX based flexible and easy-to-use access to the functionality.
* Very Good Usability – accessing the functionality with ease.

### Module Architecture

The following diagram shows the internal module design:



### Module Architecture Details

#### Presentation Layer

The Presentation Layer (Web front end) handles the requests from the searchers and generates dynamic HTML, CSS and JavaScript that is displayed in the searchers' Web browsers.

The layer it is based on the Java Web technologies and takes advantages of some AJAX techniques to make the look and fell more rich and accessible.

The Presentation Layer accesses only the Searcher Web Business Façade and cannot use directly the other layers of the module.

#### Searcher Web Business Façade

The Searcher Web Business Facade defines an interface for accessing the functionality in the Business Layer. It consists of Java interfaces and is the only part of the system that the Web front-end can access.

#### Security Authentication and Authorization Layer

The Security Authentication and Authorization Layer enforce authorization of the caller. Different business methods may have different security requirements which are configurable.

#### Business Layer

The Business Layer handles the searchers' actions. It accesses the searcher database to access the profiles of the searchers, performs search queries to the Search Engine Core and accesses the Map Server to generate maps.

The Search Engine Core and Map Server modules run as a services inside the Searcher Web's JVM and communicate directly with the other functionality through Java method calls.

#### Mock-up Implementation for Testing

The mock-up implementation is very simple implementation of the business methods defined in the Business Façade that just returns some hard-coded values and simulates the real behavior of the business layer. It allows developers to work on the frond-end modules and test them before the business layer is developed.

#### Data Access Layer

The Data Access Layer defines a set of classes to access the searcher database.

### Technologies Stack

The following technologies will be used:

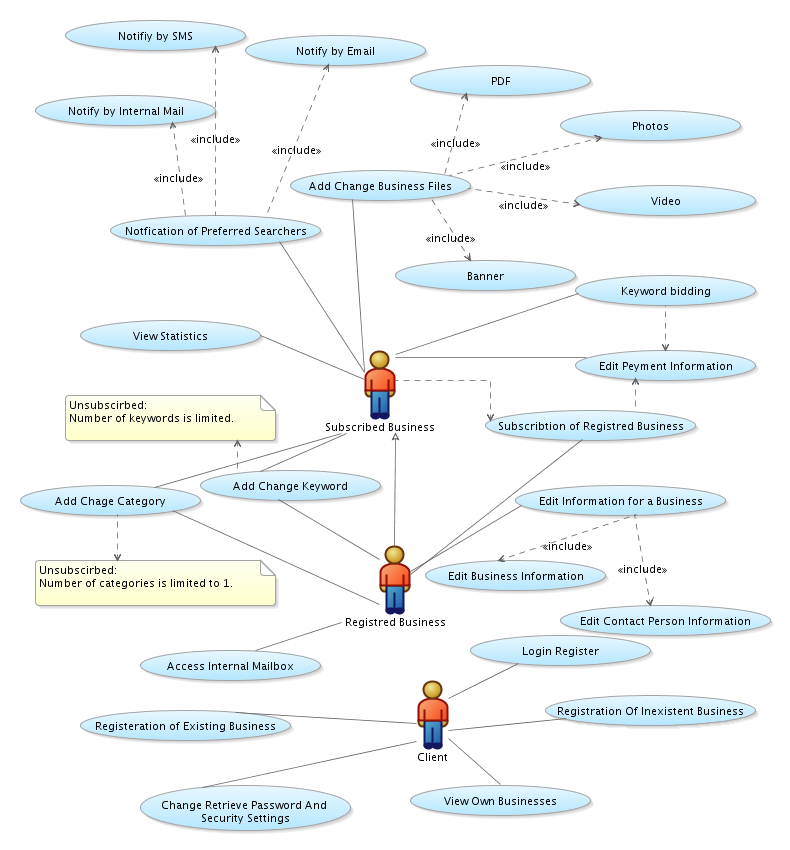
* Spring Framework – for separation between the modules interface (façade) and its implementation. This allows later the modules to be moved to separate server if required.
* JavaServer Faces (JSF) as Web application framework– for the Presentation Layer.
  + Possible frameworks:
    - JavaServer Faces (JSF) with AJAX integration
      * ICEfaces – <http://www.icefaces.org/main/demos/>
      * myFaces Tobago – <http://myfaces.apache.org/tobago/>
      * RCFaces – <http://www.rcfaces.org/starter/index.jsf>
    - Google Web Toolkit – <http://code.google.com/webtoolkit/>
    - Tapestry framework – <http://tapestry.apache.org/>
    - ZK – <http://www.zkoss.org/zkdemo/userguide/>
* Spring **JdbcTemplate** as database access framework – for communication with the database.
  + Alternatives are the following:
    - Hibernate – [http://www.hibernate.org](http://www.hibernate.org/) – very powerful and flexible
    - iBATIS – <http://ibatis.apache.org/> – high performance, less flexibility
    - Plain JDBC – will cause too much coding

## Module Business Client Account Management

The Business Client Account Management Web Front-End Module provides Web based user interface to the business clients of the search engine. Business clients are allowed to login in the module and manage their businesses.

The Business Client Account Management Module is running inside a Web server and can host other modules of the system inside its JVM.

### Use Cases



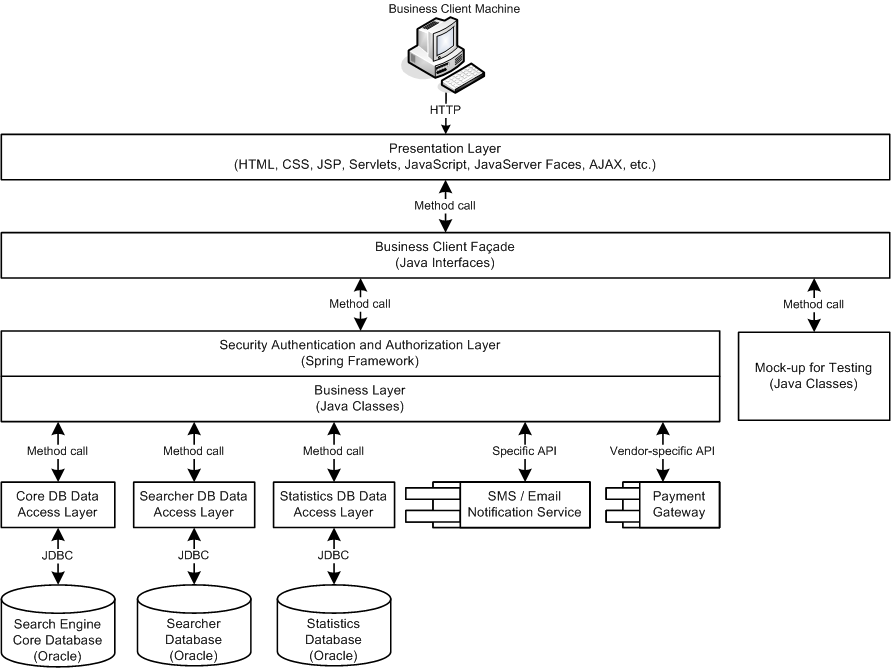
### Key Requirements and Design Considerations

The module design conforms to the following key requirements:

* User-friendly Rich Web based interface – should provide AJAX based flexible and easy-to-use access user interface to the functionality.
* High-security – secure authentication, authorization and data encryption for the financial operations.

### Module Architecture

The following diagram shows the internal module design:



### Module Architecture Details

#### Presentation Layer

The Presentation Layer (Web front-end) handles the requests from the business clients and generates dynamic HTML, CSS and JavaScript that is displayed in their Web browsers.

The Presentation Layer it is based on the Java Web technologies and takes advantages of some AJAX techniques.

The Presentation Layer accesses only the Business Client Façade and cannot use directly the other layers of the module.

#### Business Client Façade

The Business Client Facade defines an interface for accessing the functionality in the Business Layer. It consists of Java interfaces and is the only part of the system that the Web front-end can access.

#### Security Authentication and Authorization Layer

The Security Authentication and Authorization Layer enforce authorization of the caller. Different business methods may have different security requirements which are configurable.

#### Business Layer

The Business Layer handles the business clients' actions. It accesses various databases, other modules, services and remote servers to provide the requested functionality. This includes access to the Core Search Engine Database, access to the Searcher Database, access to the Statistics module, access to the SMS / E-Mail Notification module and the Payment Gateway.

The Statistics module and the SMS / E-Mail Notification modules run as services inside the JVM of the Business Account Client Web server and communicate directly with the other functionality through Java method calls.

#### Mock-up Implementation for Testing

The mock-up implementation is very simple implementation of the business methods defined in the Business Facade that just returns some hard-coded values and simulates the real behavior of the business layer. It allows developers to work on the frond-end modules and test them before the business layer is developed.

#### Core DB Data Access Layer

The Core Data Access Layer defines a set of classes to access the Search Engine Core Database.

#### Searcher DB Data Access Layer

The Searcher Data Access Layer defines a set of classes to access the Searcher Database.

#### Statistics DB Data Access Layer

The Statistics Data Access Layer defines a set of classes to access the Statistics Database.

### Technologies Stack

The following technologies will be used:

* Spring Framework – for separation between the modules interface (façade) and its implementation. This allows later the modules to be moved to separate server if required.
* JavaServer Faces (JSF) as Web application framework– for the Presentation Layer.
  + Possible frameworks:
    - JavaServer Faces (JSF) with AJAX integration
      * ICEfaces – <http://www.icefaces.org/main/demos/>
      * myFaces Tobago – <http://myfaces.apache.org/tobago/>
      * RCFaces – <http://www.rcfaces.org/starter/index.jsf>
    - Google Web Toolkit – <http://code.google.com/webtoolkit/>
    - Tapestry framework – <http://tapestry.apache.org/>
    - ZK – <http://www.zkoss.org/zkdemo/userguide/>
* Hibernate as database access framework – for communication with the database.
  + Alternatives are the following:
    - Spring **JdbcTemplate** – not enough flexible
    - iBATIS – <http://ibatis.apache.org/> – high performance, less flexibility
    - Plain JDBC – will cause too much coding

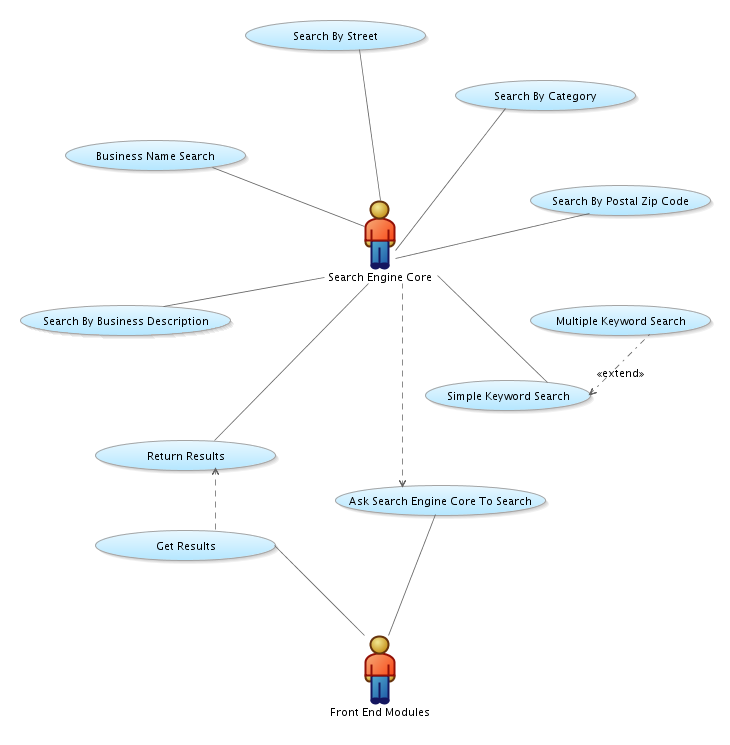
## Module Search Engine Core

The Search Engine Core module handles users search requests. It can run as a service in the JVM of the Web server running the front-end or as separate server and provides searching in the Search Engine Core database.

Users request search operations through standard Java method calls or through remote method invocation, the search engine performs searching and returns the answer to the queries as Java objects.

Due to performance considerations this module typically runs in the same JVM where the front-end runs (to reduce the communication overhead).

### Use Cases



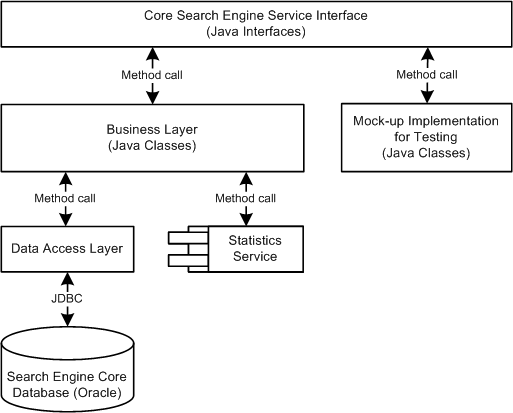
### Key Requirements and Design Considerations

This module design conforms to the following key requirements:

* Performance – should be able to handle thousands queries in the same time.
* Small Bandwidth Consumption – not too much network load for the Web front-end.
* Stateless – does not preserve state. This allows the module to be replicated in many instances located at different physical machines.
* Reliability and robustness – the module should be highly reliable and robust, with no downtime.

### Module Architecture

The following diagram shows the Search Engine Core internal module design:



### Module Architecture Details

#### Core Search Engine Service Interface

The Core Search Engine Service Interface defines an interface for querying the search engine. It consists of Java interfaces.

#### Business Layer

The Business Layer defines the implementation of the module, its business logic. It accesses the Core Search Engine Database through the Data Access Layer. It runs the Statistics Module as service in the same JVM and communicates with it by calling methods.

#### Mock-up Implementation for Testing

The mock-up implementation is very simple implementation of the business methods defined in the Service Interface that just returns some hard-coded values and simulates the real behavior of the business layer. It allows developers to work on the frond-end modules and test them before the business layer is developed.

#### Data Access Layer

The Data Access Layer defines a set of classes to access the database.

### Technologies Stack

The following technologies will be used:

* Spring Framework – for separation between the module interface and its implementation. This allows later the module to be moved to separate physical server if required.
* JDBC – for communication with the database.
* Spring **JdbcTemplate** – for simplifying database interaction.
  + Alternatives: iBATIS, plain JDBC (with no additional framework).
* Stored procedures – for encapsulating the database logic.
  + Only when necessary.

## Module Statistics

This module keeps statistics about the searches in the Search Engine. Due to performance considerations this module typically runs as service inside the Search Engine Core Module in the same JVM where it runs.

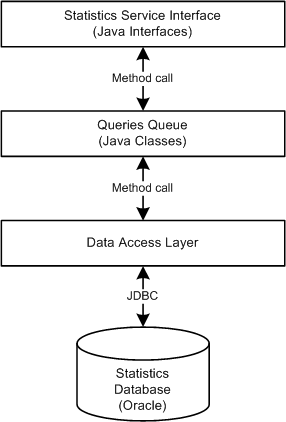
### Key Requirements and Design Considerations

The module design conforms to the following key requirements:

* Can work asynchronously – should support queue for the queries that should be analyzed for the statistics purposes and process the queue asynchronously.

### Module Architecture

The following diagram shows the internal module design:



### Module Architecture Details

#### Statistics Service Interface

The Statistics Service Interface defines an interface for adding queries to the queue for later analysis. It consists of Java interfaces.

#### Queries Queue

The Queries Queue defines the implementation of the module – its business logic. It is a queue of queries. This layer processes the queue asynchronously and updates the database statistics.

#### Data Access Layer

The Data Access Layer defines a set of classes to access the database.

### Technologies Stack

The following technologies will be used:

* Spring Framework – for separation between the modules interface (façade) and its implementation. This allows later the module to be moved to separate server if required.
* JDBC – for communication with the database.
* Spring **JdbcTemplate** – for simplifying database interaction.
* Stored procedures – for encapsulating database logic (when necessary).

## Data Import Modules

The Data Import Modules provide importing of data in the Search Engine Core Database.

There are several types data import modules.

### External Database Import Module

This kind of sub-modules transform existing relational database (e.g. MS Access DB) to the Search Engine Core Database format.

### Web Crawler Modules

These kinds of sub-modules are robots that crawl in the WWW and transform specific sites and Web directories into the Search Engine Core Database format. Different crawlers are used for the different Web sites.

### Assign Keywords Module

This sub-module of the data import modules assigns keywords based on the company profiles. It uses complex information retrieval algorithms adjusted for its needs.

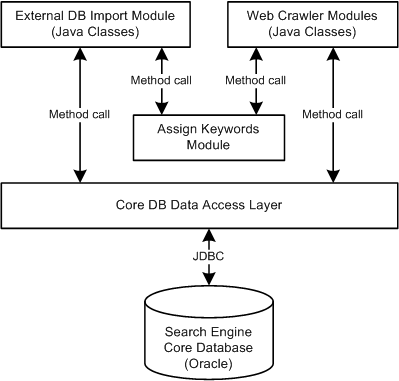
### Key Requirements and Design Considerations

The module design conforms to the following key requirements:

* High precision – the imported data should be checked for errors before importing.

### Module Architecture

The following diagram shows the internal module design:



## Module Statistics Reporting

This module will allow the search engine owners to view and analyze various the statistics and reports about the search engine, keywords, business clients and other. It will be implemented as Web application.

## Module Administration

This module will allow the system administrator to perform small administration tasks. It will be specified in more details later.

## Module Payments

This module will provide access to a payment gateway able to store securely user's payment information and to handle payment transactions. It will be specified in more details later.

## Module Mobile Access

This module will provide the searchers access to the search engine though mobile devices (WAP, SMS, MMS, etc.). It will be specified in more details later.

## Module Map Server

The map server will provide visualization of addresses on a map. It will be implemented by Google maps but this can change if better approach is found.

## Module Customer Services

This module will be front-end for the support engineers.

## Module SMS/Email Notification

This module will handle sending of emails and SMS messages and will be used as service by the other modules. It will support a queue and will process the outgoing messages asynchronously.

# Open Questions

Few open questions still remain in this technical design document:

* How to handle the maps and the map server
* How to handle the payments and the payment gateway