# Overview

Name Lookup support for assigning user names, from a modest sized collection of names, requires rapid in-memory search of name attributes. The list of names searched must occasionally be updated. Matches occur by specifying a pattern that should appear in one or more user attributes of users in the list. The document begins with a business focus on functional requirements, proceeds to more detailed descriptions in the form of user stores, and ends with a description of the solution.

# Contents

Table of Contents

[1 Overview 1](#_Toc395271762)

[2 Contents 1](#_Toc395271763)

[3 Problem Definition 2](#_Toc395271764)

[3.1 Vocabulary 2](#_Toc395271765)

[3.2 User Stories 2](#_Toc395271766)

[3.2.1 Story – Provision Searcher 2](#_Toc395271767)

[3.2.2 Story – Find with Left Match 2](#_Toc395271768)

[3.2.3 Story – Find with contains Match 3](#_Toc395271769)

[3.2.4 Story – limit matches 3](#_Toc395271770)

[3.3 Use Case Diagram 4](#_Toc395271771)

[4 Solution Description 5](#_Toc395271772)

[4.1 Key Abstractions – Static Class Diagram 5](#_Toc395271773)

[4.2 Interactions – Sequence Diagram 6](#_Toc395271774)

[5 Collaborators – Implementation Vocabulary 6](#_Toc395271775)

# Problem Definition

We detail the functional requirements and quality-of-service requirements for the search. We expect the design to be an in-memory solution. In general, multiple patterns with be sought in the user collection before the user collection changes.

## Vocabulary

|  |  |
| --- | --- |
| **Concept** | **Definition** |
| **User** | A fixed collection of searchable attributes, including first and last name along with the user Email address, which will be searched. |
| **Users** | A collection of user instances that may occasionally change (every 5 to 20 minutes). The expected maximum size is about 70,000 entries; the probable size is 2,000 entries. |
| **Pattern** | A sequence of characters that may appear in any user attribute. |
| **Match** | When the pattern is located in one or more user attributes, a match is said to occur. |
| **Left Match** | This kind of match occurs when an attribute begins with the search pattern; it is essentially a SQL “like” match. |
| **Right Match** | This kind of match occurs when an attribute contains the search pattern. |
| **Matches** | A collection, possibly limited, of users that match a pattern; the result of a search |
| **Search** | The act of invoking the searcher, provisioned with users, that results in finding the matches associated with a pattern. A search should not require, on average, more than 5 milliseconds to complete. |
| **Searcher** | The mechanism that performs the search |
| **Search User** | The entity that has access to users to be searched and will use a searcher to search those users for matches against one or more patterns. |

## User Stories

These textually describe how the search mechanism functions.

### Story – Provision Searcher

User provisioning is done in order to keep the users in the searcher up to date.

**As a** search user

The collection of users has changed, and I want the searcher to use the updated collection.

**Scenario 1**: The new collection of users is inserted into the searcher.

### Story – Find with Left Match

Left match with user attribute values

**As a** search user

I want the searcher to find matching users where one or more attributes start with the search pattern.

**Scenario 1**: Specify the pattern “Don” with left-match against the user identifiers:

. . .

Donald | Trummell | [dtrummel@gmail.com](mailto:dtrummel@gmail.com)

. . .

Robert | Frost | [d.frost@vermont.com](mailto:d.frost@vermont.com)

. . .

Daniel | Tosh | [dantosh@donbury.com](mailto:dantosh@donbury.com)

. . .

Linda | Shawndon | [linda.shawndon@ford.com](mailto:linda.shawndon@ford.com)

. . .

Le Frosh | D’onnly | [frosh@airbus.fr](mailto:frosh@airbus.fr)

. . .

**Returns**:

donald | trummell | [dtrummel@gmail.com](mailto:dtrummel@gmail.com) [lower case, left on first name]

lefrosh | donnly | [frosh@airbus.fr](mailto:frosh@airbus.fr) [lower case, left on last name]

### Story – Find with contains Match

Contains match with user attribute values

**As a** search user

I want the searcher to find matching users where attributes “contain” the search pattern.

**Scenario 1**: Specify the pattern “Don” with contains-match against the user identifiers above.

**Returns**:

donald | trummell | [dtrummel@gmail.com](mailto:dtrummel@gmail.com) [lower case, left on first name]

daniel | tosh | [dantosh@donbury.com](mailto:dantosh@donbury.com) [lower case, center on email]

linda | Shawndon | [linda.shawndon@ford.com](mailto:linda.shawndon@ford.com) [lower case, center on last name]

lefrosh | donnly | [frosh@airbus.fr](mailto:frosh@airbus.fr) [lower case, remove “’”, left on last name]

### Story – limit matches

Return the first 2 contains-match with user attribute values, leaving out the remaining matches

**As a** search user

I want the searcher to find no more than two matching users where attributes “contain” the search pattern.

**Scenario 1**: Specify the pattern “Don” with contains-match against the user identifiers above.

**Returns**:

donald | trummell | [dtrummel@gmail.com](mailto:dtrummel@gmail.com) [lower case, left on first name]

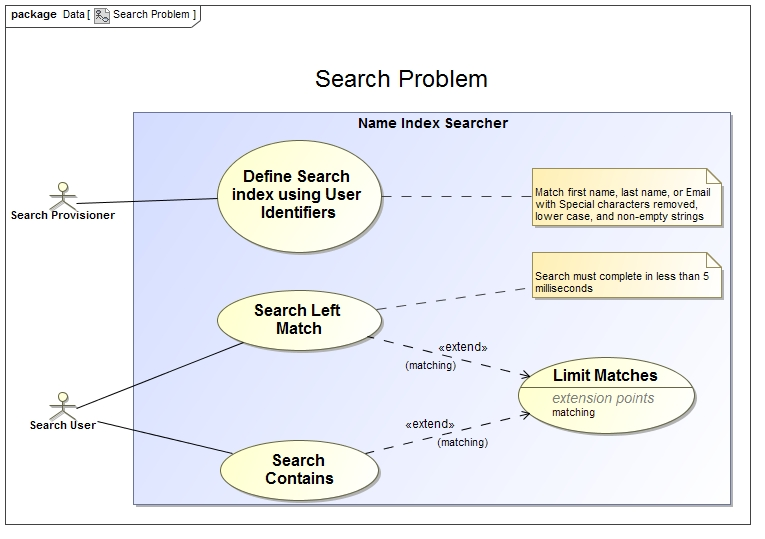
daniel | tosh | [dantosh@donbury.com](mailto:dantosh@donbury.com) [lower case, center on email]

But the searcher does NOT return:

linda | Shawndon | [linda.shawndon@ford.com](mailto:linda.shawndon@ford.com) [lower case, center on last name]

lefrosh | donnly | [frosh@airbus.fr](mailto:frosh@airbus.fr) [lower case, remove “’”, left on last name]

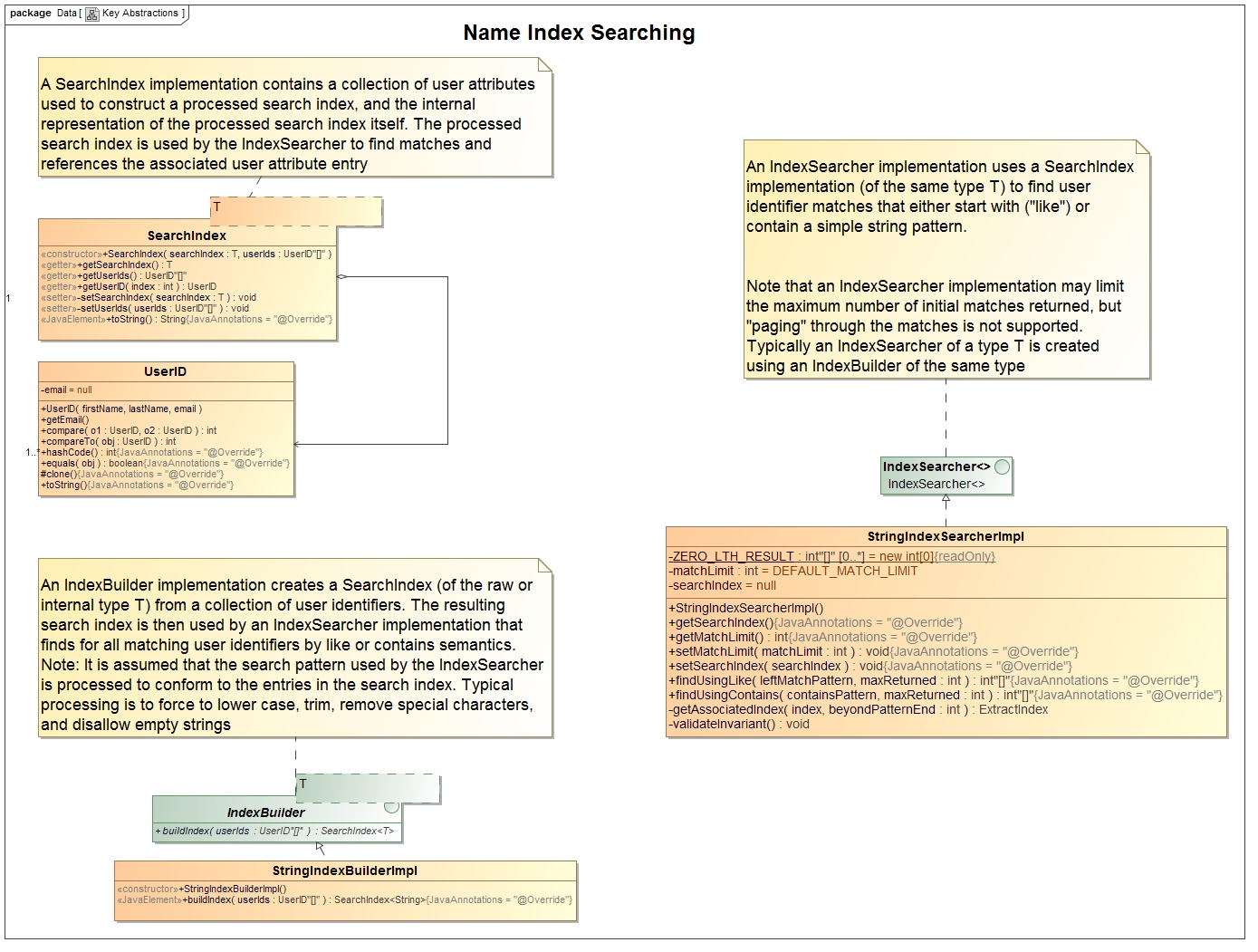
## Use Case Diagram



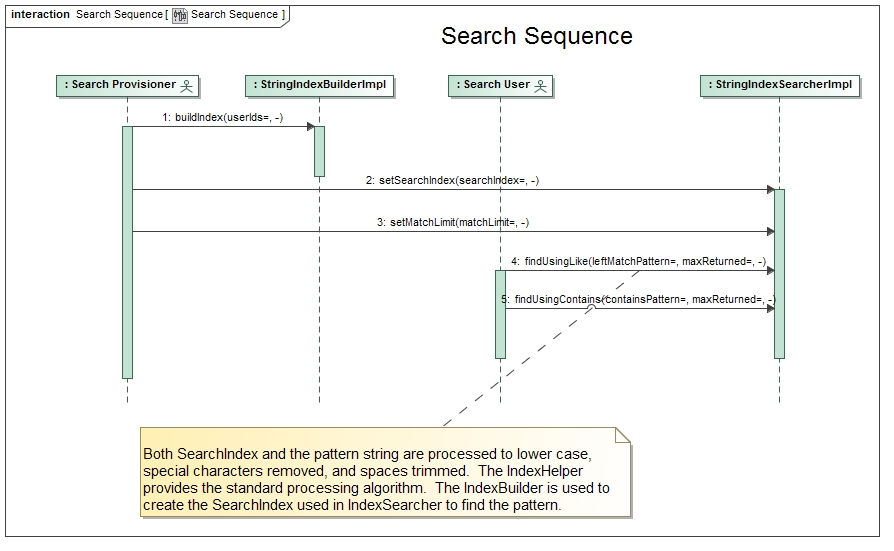
# Solution Description

UML diagrams describe the relationship between collaborators and their dynamic behavior.

## Key Abstractions – Static Class Diagram



## Interactions – Sequence Diagram



# Collaborators – Implementation Vocabulary

|  |  |  |
| --- | --- | --- |
| **Collaborator** | **Responsibilities** | **Implementation** |
| **SearchIndex**<T>  **T** is the internal or raw data type | An instance of this class contains a collection of user identifiers used to construct a processed search index, and the processed search index. The internal, or raw, data type of the processed index is used by the **IndexSearcher** to find matches and the associated user identifiers |  |
| **IndexBuilder**<T>  **T** is the internal or raw data type | An **IndexBuilder** implementation creates a **SearchIndex** (of the raw or internal type T) from a collection of user identifiers. The resulting search index is then used by an **IndexSearcher** implementation that finds for all matching user identifiers by like or contains semantics.  **Note:** It is assumed that the search pattern used by the IndexSearcher is processed to conform to the entries in the search index. Typical processing is to force to lower case, trim, remove special characters, and disallow empty strings (use **IndexHelper**) | **StringIndexBuilderImpl** |
| **IndexSearcher**<T>  **T** is the internal or raw data type | An **IndexSearcher** implementation uses a **SearchIndex** implementation (of the same type T) to find user identifier matches that either start with ("like") or contain a pattern.  Note that an **IndexSearcher** implementation may limit the maximum number of initial matches returned, but "paging" through the matches is not supported. Typically an **IndexSearcher** is created using an **IndexBuilder** | **StringIndexSearcherImpl** |
|  |  |  |
| **UserID** | Adds the Email as part of a Human identification by extending **FLName**. |  |
| **FLName** | This class manages the simple Western European notion of a person's name consisting of two parts: a first or given name, and last or family name. It supports three sets of responsibilities:   * Supplies an ordering by implementing Comparable and Comparator * Handle equals, hashCode, and toString * Provides clone |  |
| **Testing** | **AllTests** – suite with all test cases | **IndexRunner** - Test program is used to determine elapsed time various kinds of searches. The test search data comes from a **NameBuilder** – an instance that has more than 46,000 fake entries. |

▄