GSRS Architecture Overview

This document describes the design of the Global Substance Registration System (GSRS) from various architectural viewpoints.

What is GSRS?

GSRS, a web-based repository and browser for chemical substance data, is an open-source application built using the industry-standard Model-View-Controller (MVC) pattern. This project was born as a collaboration project between the U.S. National Institutes of Health (NIH) and the U.S. Food and Drug Administration (FDA), and it quickly gained adoption from a handful of enterprises which set up their own instance of it to support their business.

The GSRS code is hosted entirely on GitHub, while some compiled modules are hosted on Maven Central. In the spirit of open-source software development, each adopter of GSRS – a.k.a. collaborator – could contribute new functions to the core of the project.

GSRS uses a microservice-based architecture which allows collaborators to develop custom modules which serve to integrate GSRS with their own existing IT systems. As these modules are centrally stored in the GitHub repository, it is possible for other collaborators to use them and enhance them.

GSRS serves as a one-stop central repository for information related to chemical substances. For example, the FDA instance of GSRS serves as a central informational resource for all substances that are intended for human consumption. Other enterprises throughout the world use it as a repository for substances relevant to their business. GSRS facilitates the exchange (export and import) of substance records between different instances set up at various sites.

The first instances of GSRS were deployed by the United States government at FDA and at NIH’s National Center for Advancing Translational Sciences (NCATS).

Microservice-level GSRS architecture

We first describe the architecture of GSRS at the microservice level. GSRS is a web application which employs the microservice architecture. Figure 1 shows this architecture. Each entity in the diagram represents a microservice.

Each of the major domains within GSRS (e.g. Substances, Applications, Adverse Events, Clinical Trials, Impurities, Products and SSG4M) has two parts:

* 1. A module or library of domain-specific functionality
  2. A microservice that makes the functionality available to calling programs using an API.

Each microservice (a.k.a. “service”) exposes an Application Programming Interface (API) which is accessible using the JSON-based RESTful protocol (a.k.a. REST). Microservices communicate between them exclusively by accessing each other’s REST interface. Messages are serialized and encoded in JSON.

The Substances module is the core GSRS module. Applications, Adverse Events, Clinical Trials, Impurities, Products and SSG4m are the GSRS extensions built by FDA and for FDA, but they can be customized for use at any enterprise that integrates GSRS within its IT infrastructure. A minimal GSRS system includes the User Interface (UI) service shown at the top of the diagram (a.k.a. the Frontend service), the Gateway (which routes requests from the UI to the other services), and the Substance service. Additional extensions can be integrated, customized, and deployed as needed. New customized extensions can be developed as well. The optional Discovery server microservice can be used to register and find microservices. As of this writing in March 2024, NCATS and FDA systems do not use the Discovery server microservice.

Diagram

Description automatically generated

Figure 1: Microservice level architecture

# A closer look at the Substances microservice

The main functionalities of the Substances microservice are:

* Data load and Import
* Data export
* Chemical rendering
* Image to structure conversion (using the NCATS MolVec library)
* User authentication
* Structure search
* Text search
* Sequence search
* Substance validation
* Reporting
* Approval ID generation

The backend of GSRS uses the Spring Boot framework. For data storage, it uses Apache Lucene indexes hosted on the application server and a relational database hosted on a database server. Currently supported database flavors are Oracle, PostgreSQL, MySQL and MariaDB.

Below is a diagram of the basic components of the Substance microservice. The Frontend service interacts with the backend through the Gateway. The backend is developed using the Java Spring Boot framework. The business logic part is responsible for dealing with the functionalities. The Controller-Service-Repository pattern is used heavily in the code base. Spring Data JPA (Java Persistence API) and Entity Manager are both used to interact with databases. Data stored in the database is also maintained in Lucene indexes on the application host for fast retrieval by the application. GSRS provides functionality (through its Schedule Tasks in admin panel) to rebuild the Lucene indexes from the database table data if for any reason the indexes become stale.   
Diagram

Description automatically generated  
 Figure 2: The Core Components of GSRS

Code-level Architecture

The GSRS Starter library defines basic and common operations. It also provides interfaces, abstract classes, and extendable classes that other services can build upon. The Substances module and other entities extend and define specialized functionality applicable to them.Diagram

Description automatically generated

Figure 3: GSRS Entity Dependencies

# GSRS Technical Stack

**GSRS v3.0 uses the following technologies:**

* Java (8 ,11 or 17. FDA is using JDK 11)
* **Spring Boot** framework
* **Angular**, an advanced JavaScript-based framework for development of the user interface
* Apache **Maven** for building the code from pre-compiled pieces
* RESTful API protocol for communication between the GSRS front-end, gateway, backend and other GSRS microservices
* Hibernate ORM, Hibernate Java Persistence API (JPA), for communication between the GSRS backend and the GSRS databases
* Apache **Lucene** (open-source indexing and search software)
* Scilligence **JSDraw** (licensed) or the open-source **CDK** (open source) for drawing and viewing chemical diagram

**NCATS/community custom science libraries:**

* MolVec (optical recognition of chemical structure drawings, developed by Tyler and Danny at NCATS)
* MolWitch (a Bridge interface to abstract the underlying cheminformatics library is used, developed by Danny at NCATS)
* Jillion (bioinformatics library for genomic data used to read sequence data, which provides the alignment part of the sequence search in GSRS)
* Chemistry Development Kit (CDK, written in the course of the past 20 years through the contribution of many people including Danny of NCATS)

**Data management:**

* An Excel suite of tools custom-written by FDA (using Microsoft .Net) which interfaces with the GSRS RESTful API to create/Insert, update, or delete substance data
* Toad v12 for Oracle data management

Below is a diagram of the GSRS technology stack. The Frontend is based on the Angular framework. The Substance entity service and the core framework are in the backend.

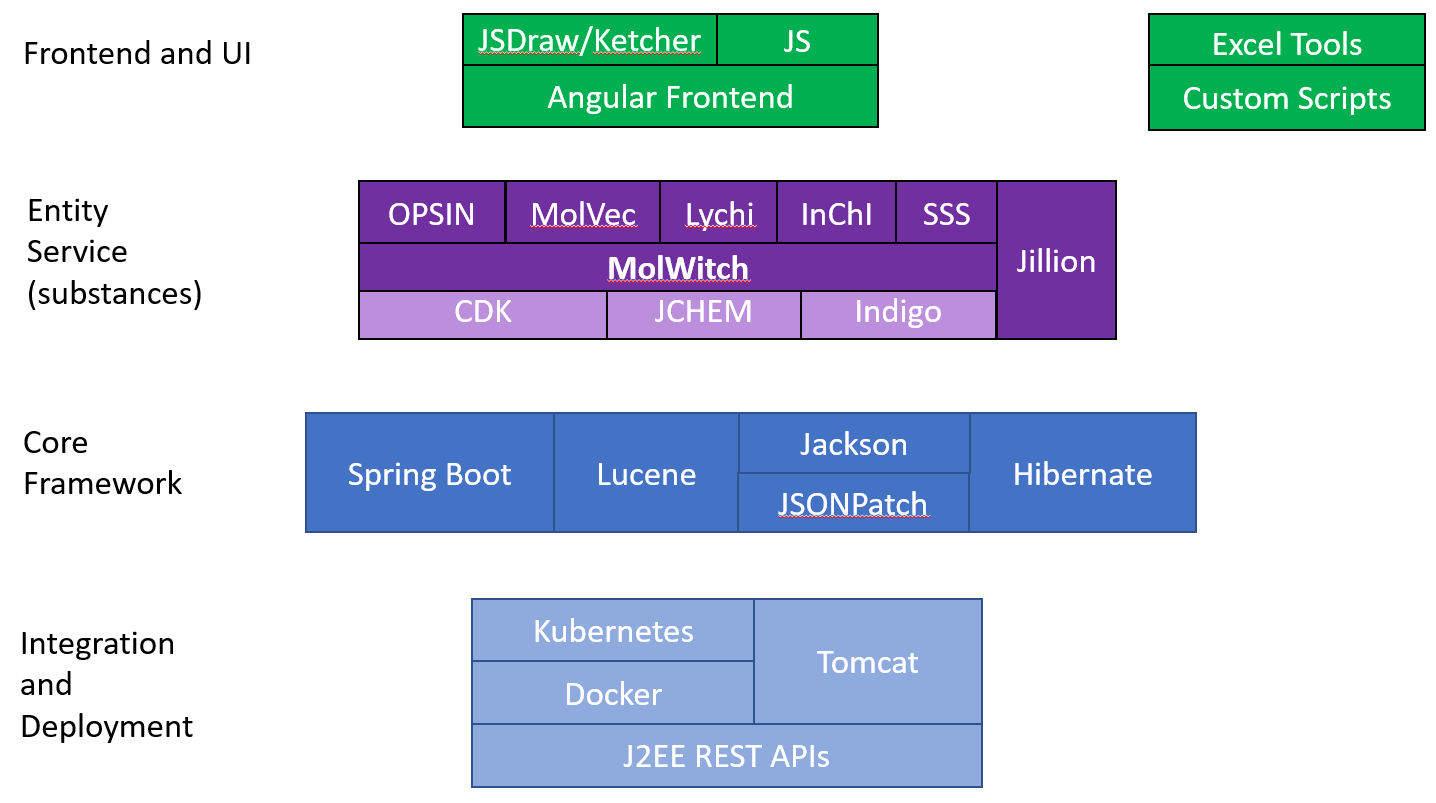


Figure 4: GSRS Tech Stack

Notes:

JSDraw is a chemical structure drawing package from Scilligence.

Ketcher is a chemical structure drawing package from EPAM.

The GSRS Excel Tools are an NCATS add-in for Microsoft Excel which provide access to viewing, uploading, or modifying GSRS data via the REST API.

Custom scripts are any type of program, either scripts or a more substantial programming environment, that access the GSRS API.

OPSIN: (Open Parser for Systematic IUPAC nomenclature) is a software library from the University of Cambridge that generate chemical structures from IUPAC names.

MolVec is an NCATS-developed library that optically recognizes chemical structures from graphical depictions in digital images.

MolWitch is an NCATS-developed library for the handling of chemical structures. It wraps an underlying chemical toolkit, so operations are expressed in a general and consistent way.

SSS is a chemical substructure search tool.

LyChI is a chemical identifier generated from the structure. It was developed by NCATS.

CDK is an open-source chemical toolkit, available on GitHub.

Jillion is a Java framework from the J. Craig Venter Institute, for reading, writing, analyzing and manipulating sequence and assembly data.

The other terms are standard technology and can be found using a web search engine.

# Configuration layers and implement/deployment layers

GSRS is designed to meet different needs and requirements. It can be easily reconfigured or extended to accommodate future and uncertain features and functionalities, as well as the different requirements from various organizations.

The graph below shows the different layers of customizations users could implement. From the bottom, GSRS Spring Boot Starter provides abstract classes, interfaces and basic and common functionalities. Developers can extend and implement these abstract classes and interfaces and build upon them. GSRS have defined entity classes and libraries, such as Substances, Codes, Names and References. Users can use these or define their own entity classes and libraries. Users can select Entities or add new Entities. GSRS can use Oracle, PostgreSQL, MariaDB or MySQL as data source. For deployment, users also have the option to use standalone Tomcat or use single Docker container or multiple Docker containers to deploy GSRS system. GSRS have config files for Substances, Gateway, Frontend, and config files to config functionalities in gsrs-core.conf and substances-core.conf. From top to bottom, it gets more difficult to understand and make changes, but the changes will have more impact to the system.

Every extension has its own config file.

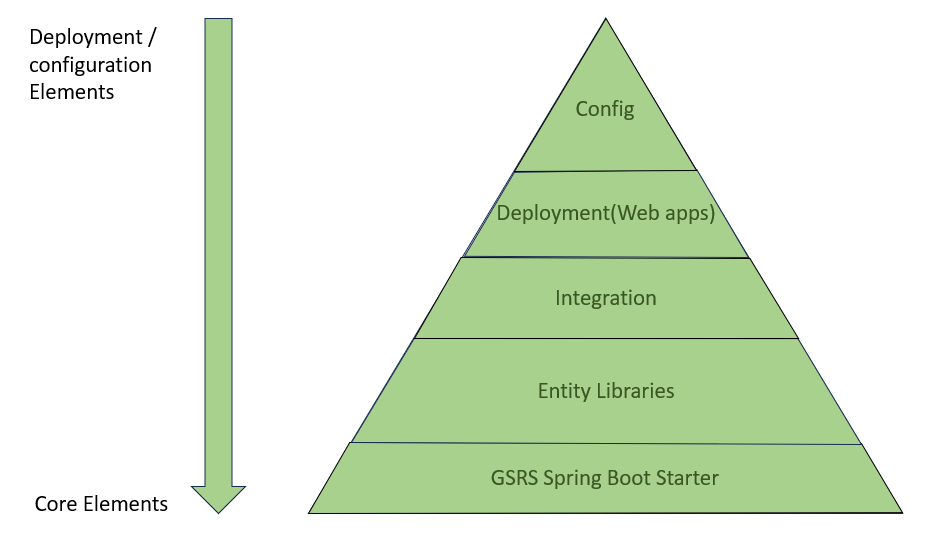


Figure 5: Configurable Layers

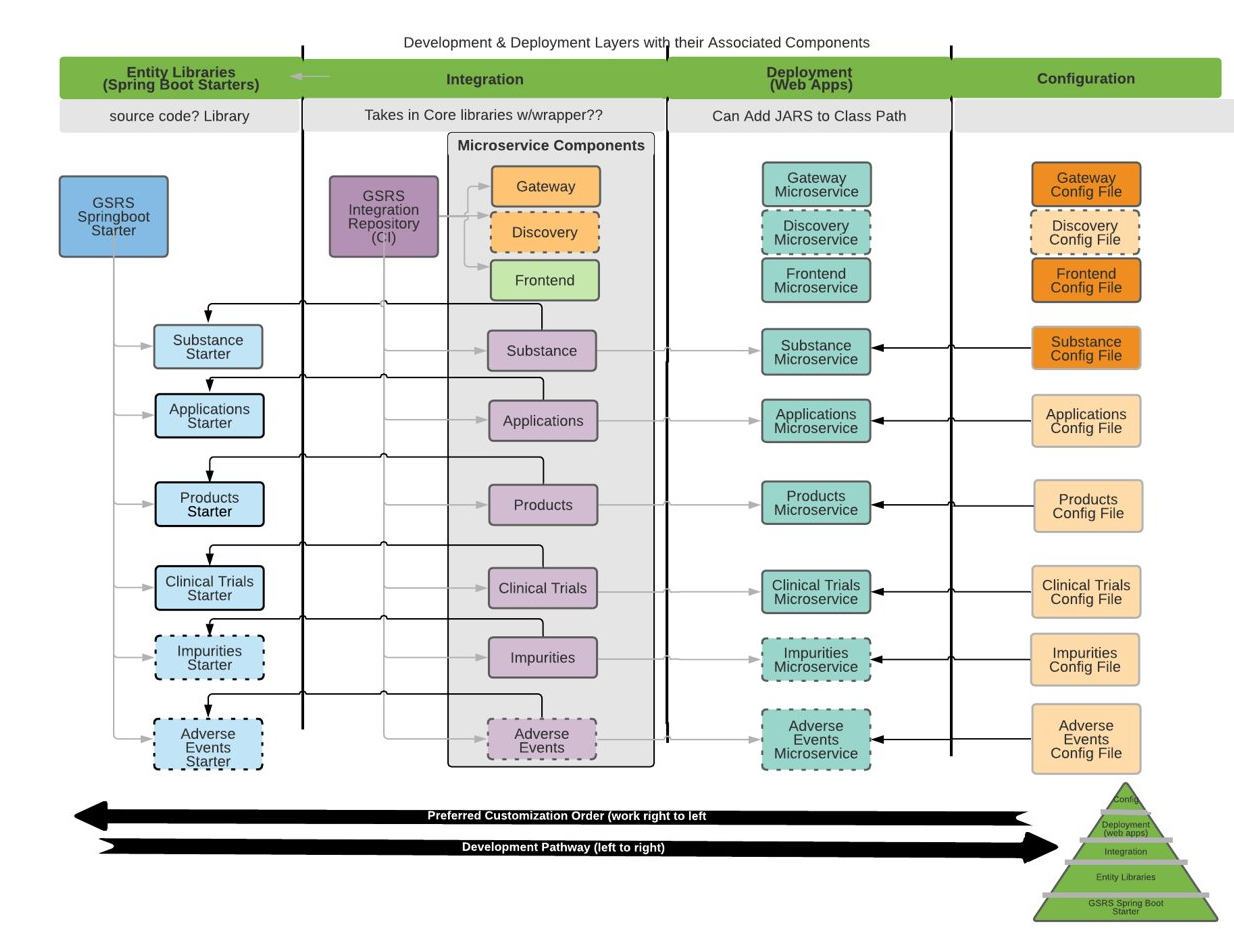


Figure 6: Development & Deployment Layers with Components

# Substances and extensions

GSRS can be extended and customized in the following ways:

* Custom Index Value Makers
* Custom Exporters
* Custom Validation Rules
* Custom Structure Hashing
* Custom Entity Processors
* Custom Scheduled Tasks
* Custom Substance Hierarchy
* Custom GSRS Microservice

Developers can implement the IndexValueMaker interface and define their own index value makers for index fields, values, facets and suggestions. Similarly, they can define new exporters, validation rules, structure hashing all the way to new microservices.

Each GSRS microservice extension has its own indexes and databases. Extensions do need access to the substance database to a few tables. This is depicted in the graph below. The top blue items depict the Substances module with its own database and Lucene indexes. The bottom green items are a sample GSRS extension, in this case the FDA Clinical Trials service. It has its own database and Lucene indexes. It also connects to the Substance service’s database.

Diagram

Description automatically generatedFigure 5: GSRS Extensions’ communication with the core GSRS database

Each extension has dependencies on the GSRS Starter and the GSRS Substance modules, and this is defined in the extension’s pom.xml file.

# Data Dictionary

On GSRS download website, you will find out with each public release comes with a data dictionary.

Data dictionary is a Spreadsheet explaining for each field:

* Describes what it means
* Data Type
* Which Entity it belongs to
* Where in the entity that field exists
* The index path to that field
* Covers Substances, and FDA extensions   
   (Applications, Products, Clinical Trials ...)

Note that the data model in the backend GSRS Substances database is different from that of the data dictionary. The data dictionary shows our conceptual data model – what are the types of data we handle for Substances. That's how a user thinks about a substance in GSRS: it has one or more names, zero to many codes, possibly properties, etc. The tables in the backend GSRS Substances database show a concrete storage structure for this data. Almost everything within conceptual model maps to the tables but there may be transformations.