Jmix Documentation
Introduction
What's New
Setup
Tutorial
Using Studio
Data Model
Data Access
User Interface
Localization
Managing Files
Security
Other Features
Modularity and Extension
Testing
Deployment
Add-ons
Appendices

Contents
What is Jmix?
Getting Started
Architecture Overview
Containers
Components

Introduction

Welcome to the Jmix documentation!

What is Jmix?

Jmix is a high-level framework for enterprise web applications. It comes with advanced tooling and a rich set of functional modules.

Jmix is best suited for creating data-centric applications with a complex data model and rich user interface. It is a full-stack framework in the sense that it provides support for creating both the backend and UI of your application.

Jmix is based on Spring Boot, which is a de-facto standard for creating enterprise Java web applications. It means that you can use a lot of third-party libraries and frameworks with minimal configuration, in addition to the functionality provided by Jmix.

Jmix Studio is a plugin for IntelliJ IDEA that helps you at all stages of the application development: creating and configuring a project, defining data model, generating database migration scripts, developing UI views in a visual editor. It provides advanced navigation, code completion and inspections specific to Jmix projects.

Getting Started

If you are new to Jmix, check out the Jmix Introduction for Developers video.

Complete the Tutorial to learn basic concepts and techniques of programming with Jmix.

If you want to get a high-level idea of what Jmix is before diving into code, see the architecture overview below.

The documentation contains everything you need to know for developing with Jmix:

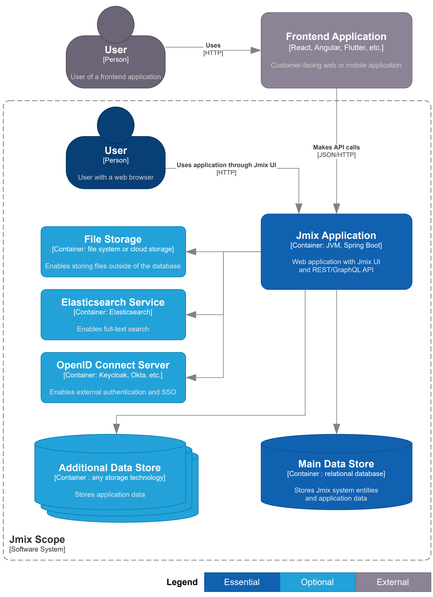
- Using Studio section gives you detailed information about Jmix Studio features and installation options.
- The top-level sections from Data Model to Deployment explain basic topics that help you develop any kind of application.
- The Add-ons section describes additional modules that can be used in your project.

Architecture Overview

Here you can find a few diagrams in C4 Model notation that show a birds eye view of the Jmix landscape.

Containers

First, let's divide a typical information system built with Jmix into containers. A container here is a separately runnable/deployable unit that executes code or stores data.



As you can see, the main part of the system includes a web application running on JVM and a relational database. The web application is based on Spring Boot and can be written in Java or Kotlin. A relational database is used to store data of Jmix subsystems (for example, security configuration) and as a main storage of application data.

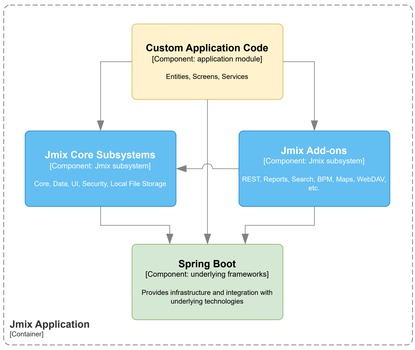
The Jmix application can connect to any number of additional data stores and use a separate file storage on a file system or in the cloud. Also, Jmix offers integrations with Elasticsearch for full-text search through the application data, and with an OIDC server for external authentication and SSO.

The Jmix application provides users with a web interface. It's created in Java/Kotlin and XML using the Jmix UI subsystem.

You can also create a separate frontend application and connect it with the Jmix backend through the Jmix REST API or by creating custom endpoints. Note that frontend applications are out of Jmix scope, you can create them with any technology of your choice.

Components

Let's zoom in to the Jmix Application container and explore its components.



A Jmix application always contains dependencies to Spring Boot and to a number of Jmix subsystems. There are two categories of Jmix subsystems:

- *Core subsystems* provide system-level functionality and are included in most projects.
- *Add-ons* are optional subsystems that can be included in the project on demand. The add-ons are published at Jmix marketplace.

Technically, both core subsystems and add-ons are organized in the same way. A subsystem contains one or more functional modules and corresponding Spring Boot starters, each packaged in a JAR file. A subsystem can depend on other subsystems: for example, many add-ons contain UI views that require dependency on the core UI subsystem.

Next
What's New ›