Continue

JasperReports is an open source reporting library that enables users to create pixel-perfect reports that can be printed or exported in many formats including PDF, HTML, and XLS. In this article, we'll explore its key features and classes, and implement examples to showcase its capabilities. 2. Maven Dependency First, we need to add the jasperreports dependency to our pom.xml: net.sf.jasperreports jasperreports 6.20.0 The latest version of this artifact can be found here. 3. Report Templates Report designs are defined in JRXML files. These are ordinary XML files with a particular structure that JasperReports engine can interpret. Let's now have a look at only the relevant structure of the JRXML files – to understand better the Java part of the report generation process, which is our primary focus. Let's create a simple report to show employee information: 3.1. Compiling Reports JRXML files need to be compiled so the report engine can fill them with data. Let's perform this operation with the help of the JasperCompilerManager class: InputStream employeeReportStream = getClass().getResourceAsStream("/employeeReport.jrxml"); JasperReport jasperReport = JasperCompileManager.compileReport(employeeReportStream); To avoid compiling it every time, we can save it to a file: JRSaver.saveObject(jasperReport, "employeeReport.jasper"); 4. Populating Reports The most common way to fill compiled reports is with records from a database. This requires the report to contain a SQL query the engine will execute to obtain the data. First, let's modify our report to add a SQL query: ... Now, let's create a simple data source: @Bean public DataSource dataSource() { return new EmbeddedDatabaseBuilder() .setType(EmbeddedDatabaseType.HSQL) .addScript("classpath:employee-schema.sql") .build(); } Now, we can fill the report: JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport, null, dataSource.getConnection()); Note that we are passing null to the second argument since our report doesn't receive any parameters yet. 4.1. Parameters Parameters are useful for passing data to the report engine that it can not find in its data source or when data changes depending on different runtime conditions. We can also change portions or even the entire SQL query with parameters received in the report filling operation. First, let's modify the report to receive three parameters: // ... Now, let's add a title section to show the title parameter: // ... ... Next, let's alter the query to use the minSalary and condition parameters: SELECT * FROM EMPLOYEE WHERE SALARY >= $P{minSalary} AND $P!{condition} Note the different syntax when using the condition parameter. This tells the engine that the parameter should not be used as a standard PreparedStatement parameter, but as if the value of that parameter would have been written originally in the SQL query. Finally, let's prepare the parameters and fill the report: Map parameters = new HashMap(); parameters.put("title", "Employee Report"); parameters.put("minSalary", 15000.0); parameters.put("condition", " LAST_NAME ='Smith' ORDER BY FIRST_NAME"); JasperPrint jasperPrint = JasperFillManager.fillReport(..., parameters, ...); Note that the keys of parameters correspond to parameter names in the report. If the engine detects a parameter is missing, it will obtain the value from defaultValueExpression of the parameter if any. 5. Exporting To export a report, first, we instantiate an object of an exporter class that matches the file format we need. Then, we set our previous filled report as input and define where to output the resulting file. Optionally, we can set corresponding report and export configuration objects to customize the exporting process. 5.1. PDF JRPdfExporter exporter = new JRPdfExporter(); exporter.setExporterInput(new SimpleExporterInput(jasperPrint)); exporter.setExporterOutput(new SimpleOutputStreamExporterOutput("employeeReport.pdf")); SimplePdfReportConfiguration reportConfig = new SimplePdfReportConfiguration(); reportConfig.setSizePageToContent(true); reportConfig.setForceLineBreakPolicy(false); SimplePdfExporterConfiguration exportConfig = new SimplePdfExporterConfiguration(); exportConfig.setMetadataAuthor("baeldung"); exportConfig.setEncrypted(true); exportConfig.setAllowedPermissionsHint("PRINTING"); exporter.setConfiguration(reportConfig); exporter.setConfiguration(exportConfig); exporter.exportReport(); JRXlsxExporter exporter = new JRXlsxExporter(); // Set input and output ... SimpleXlsxReportConfiguration reportConfig = new SimpleXlsxReportConfiguration(); reportConfig.setSheetNames(new String[] { "Employee Data" }); exporter.setConfiguration(reportConfig); exporter.exportReport(); JRCsvExporter exporter = new JRCsvExporter(); // Set input ... exporter.setExporterOutput(new SimpleWriterExporterOutput("employeeReport.csv")); exporter.exportReport(); HtmlExporter exporter = new HtmlExporter(); // Set input ... exporter.setExporterOutput(new SimpleHtmlExporterOutput("employeeReport.html")); exporter.exportReport(); Subreports are nothing more than a standard report embedded in another report. First, let's create a report to show the emails of an employee: Now, let's modify our employee report to include the previous one: Note that we are referencing the subreport by the name of the compiled file and passing it the idEmployee and current report connection as parameters. Next, let's compile both reports: InputStream employeeReportStream = getClass().getResourceAsStream("/employeeReport.jrxml"); JasperReport jasperReport = JasperCompileManager.compileReport(employeeReportStream); JRSaver.saveObject(jasperReport, "employeeReport.jasper"); InputStream emailReportStream = getClass().getResourceAsStream("/employeeEmailReport.jrxml"); JRSaver.saveObject(JasperCompileManager.compileReport(emailReportStream), "employeeEmailReport.jasper"); Our code for filling and exporting the report doesn't require modifications. 7. Conditional Display With printWhenExpression In addition, we can use printWhenExpression to conditionally display report elements based on certain criteria. This means that elements like text fields, images, and bands can be shown or hidden dynamically according to the data or parameters in our report. Below is an example of how to modify the JRXML file to include a null check using printWhenExpression. We'll check for non-null values in the FIRST_NAME, LAST_NAME, and SALARY fields before rendering the content in the detail band: This expression ensures that the entire content of the band will only be displayed if all these fields have valid (non-null) values. After updating the JRXML file, we need to compile and fill the report as before. 8. Conclusion In this article, we had a brief look at the core features of the JasperReports library. We were able to compile and populate reports with records from a database; we passed parameters to change the data shown in the report according to different runtime conditions, embedded subreports and exported them to the most common formats. The code backing this article is available on GitHub. Once you're logged in as a Baeldung Pro Member, start learning and coding on the project. Computer software installed on multiple computing platforms "Cross-platform" redirects here. For the railway station interchange, see cross-platform interchange. For the game term, see cross-platform play. "Multi-platform" redirects here. For the mode of storytelling in television, see multi-platform television. Within computing, cross-platform software (also called multi-platform software, platform-agnostic software, or platform-independent software) is computer software that is designed to work in several computing platforms.[1] Some cross-platform software requires a separate build for each platform, but some can be directly run on any platform without special preparation, being written in an interpreted language or compiled to portable bytecode for which the interpreters or run-time packages are common or standard components of all supported platforms.[2] For example, a cross-platform application may run on Linux, macOS and Microsoft Windows. Cross-platform software may run on many platforms, or as few as two. Some frameworks for cross-platform development are Codename One, ArkUI-X, Kivy, Qt, GTK, Flutter, NativeScript, Xamarin, Apache Cordova, Ionic, and React Native.[3] Main article: Computing platform Platform can refer to the type of processor (CPU) or other hardware on which an operating system (OS) or application runs, the type of OS, or a combination of the two.[4] An example of a common platform is Android which runs on the ARM architecture family. Other well-known platforms are Linux/Unix, macOS and Windows, these are all cross-platform.[4] Applications can be written to depend on the features of a particular platform—either the hardware, OS, or virtual machine (VM) it runs on. For example, the Java platform is a common VM platform which runs on many OSs and hardware types. A hardware platform can refer to an instruction set architecture. For example: ARM or the x86 architecture. These machines can run different operating systems. Smartphones and tablets generally run ARM architecture, these often run Android or iOS and other mobile operating systems. A software platform can be either an operating system (OS) or programming environment, though more commonly it is a combination of both. An exception is Java, which uses an OS-independent virtual machine (VM) to execute Java bytecode. Some software platforms are: Android (ARM64) ChromeOS (ARM32, ARM64, IA-32, x86-64) Common Language Infrastructure (CLI) by Microsoft, implemented in: The legacy .NET Framework that works only on Microsoft Windows. The newer .NET framework (simply called ".NET") that works across Microsoft Windows, macOS, and Linux. Other implementations such as Mono (formerly by Novell and Xamarin[5]) HarmonyOS (ARM64, RISC-V, x86, x64, and LoongArch) iOS ((ARMv8-A)) iPadOS (ARMv8-A) Java Linux (Alpha, ARC, ARM, C-Sky, Hexagon, LoongArch, m68k, Microblaze, MIPS, Nios II, OpenRISC, PA-RISC, PowerPC, RISC-V, s390, SuperH, SPARC, x86, Xtensa) macOS x86, ARM (Apple silicon) Microsoft Windows (IA-32, x86-64, ARM, ARM64) PlayStation 4 (x86), PlayStation 3 (PowerPC) and PlayStation Vita (ARM) Solaris (SPARC, x86) SPARC Unix (many platforms since 1969) Web browsers – mostly compatible with each other, running JavaScript web-apps Xbox Minor, historical AmigaOS (m68k), AmigaOS 4 (PowerPC), AROS (x86, PowerPC, m68k), MorphOS (PowerPC) Atari TOS, MiNT BSD (many platforms; see NetBSDnet,[clarification needed] for example) DOS-type systems on the x86: MS-DOS, PC DOS, DR-DOS, FreeDOS OS/2, eComStation BeOS (PowerPC, x86) Main article: Java (software platform) The Java language is typically compiled to run on a VM that is part of the Java platform. The Java virtual machine (Java VM, JVM) is a CPU implemented in software, which runs all Java code. This enables the same code to run on all systems that implement a JVM. Java software can be executed by a hardware-based Java processor. This is used mostly in embedded systems. Java code running in the JVM has access to OS-related services, like disk input/output (I/O) and network access, if the appropriate privileges are granted. The JVM makes the system calls on behalf of the Java application. This lets users to decide the appropriate protection level, depending on an access-control list (ACL). For example, disk and network access is usually enabled for desktop applications, but not for browser-based applets. The Java Native Interface (JNI) can also be used to access OS-specific functions, with a loss of portability. Currently, Java Standard Edition software can run on Microsoft Windows, macOS, several Unix-like OSs, and several real-time operating systems for embedded devices. For mobile applications, browser plugins are used for Windows and Mac based devices, and Android has built-in support for Java. There are also subsets of Java, such as Java Card or Java Platform, Micro Edition, designed for resource-constrained devices. For software to be considered cross-platform, it must function on more than one computer architecture or OS. Developing such software can be a time-consuming task because different OSs have different application programming interfaces (API). Software written for one OS may not automatically work on all architectures that OS supports. Just because software is written in a popular programming language such as C or C++, it does not mean it will run on all OSs that support that language—or even on different versions of the same OS. Web applications are typically described as cross-platform because, ideally, they are accessible from any web browser: the browser is the platform. Web applications generally employ a client–server model, but vary widely in complexity and functionality. It can be hard to reconcile the desire for features with the need for compatibility. Basic web applications perform all or most processing from a stateless server, and pass the result to the client web browser. All user interaction with the application consists of simple exchanges of data requests and server responses. This type of application was the norm in the early phases of World Wide Web application development. Such applications follow a simple transaction model, identical to that of serving static web pages. Today, they are still relatively common, especially where cross-platform compatibility and simplicity are deemed more critical than advanced functionality. Prominent examples of advanced web applications include the Web interface to Gmail and Google Maps. Such applications routinely depend on additional features found only in the more recent versions of popular web browsers. These features include Ajax, JavaScript, Dynamic HTML, SVG, and other components of rich web applications. Because of the competing interests of compatibility and functionality, numerous design strategies have emerged. Many software systems use a layered architecture where platform-dependent code is restricted to the upper- and lowermost layers. Graceful degradation attempts to provide the same or similar functionality to all users and platforms, while diminishing that functionality to a least common denominator for more limited client browsers. For example, a user attempting to use a limited-feature browser to access Gmail may notice that Gmail switches to basic mode, with reduced functionality but still of use. Some software is maintained in distinct codebases for different (hardware and OS) platforms, with equivalent functionality. This requires more effort to maintain the code, but can be worthwhile where the amount of platform-specific code is high. This strategy relies on having one codebase that may be compiled to multiple platform-specific formats. One technique is conditional compilation. With this technique, code that is common to all platforms is not repeated. Blocks of code that are only relevant to certain platforms are made conditional, so that they are only interpreted or compiled when needed. Another technique is separation of functionality, which disables functionality not supported by browsers or OSs, while still delivering a complete application to the user. (See also: Separation of concerns.) This technique is used in web development where interpreted code (as in scripting languages) can query the platform it is running on to execute different blocks conditionally.[6] Third-party libraries attempt to simplify cross-platform capability by hiding the complexities of client differentiation behind a single, unified API, at the expense of vendor lock-in. Responsive web design (RWD) is a Web design approach aimed at crafting the visual layout of sites to provide an optimal viewing experience—easy reading and navigation with a minimum of resizing, panning, and scrolling—across a wide range of devices, from mobile phones to desktop computer monitors. Little or no platform-specific code is used with this technique. Cross-platform applications need much more integration testing. Some web browsers prohibit installation of different versions on the same machine. There are several approaches used to target multiple platforms, but all of them result in software that requires substantial manual effort for testing and maintenance.[7] Techniques such as full virtualization are sometimes used as a workaround for this problem. Tools such as the Page Object Model allow cross-platform tests to be scripted so that one test case covers multiple versions of an app. If different versions have similar user interfaces, all can be tested with one test case. Web applications are becoming increasingly popular but many computer users still use traditional application software which does not rely on a client/web-server architecture. The distinction between traditional and web applications is not always clear. Features, installation methods and architectures for web and traditional applications overlap and blur the distinction. Nevertheless, this simplifying distinction is a common and useful generalization. Traditional application software has been distributed as binary files, especially executable files. Executables only support the platform they were built for—which means that a single cross-platform executable could be very bloated with code that never executes on a particular platform. Instead, generally there is a selection of executables, each built for one platform. For software that is distributed as a binary executable, such as that written in C or C++, there must be a software build for each platform, using a toolset that translates—transcompiles—a single codebase into multiple binary executables. For example, Firefox, an open-source web browser, is available on Windows, macOS (both PowerPC and x86 through what Apple Inc. calls a Universal binary), Linux, and BSD on multiple computer architectures. The four platforms (in this case, Windows, macOS, Linux, and BSD) are separate executable distributions, although they come largely from the same source code. In rare cases, executable code built for several platforms is combined into a single executable file called a fat binary. The use of different toolsets may not be enough to build a working executables for different platforms. In this case, programmers must port the source code to the new platform. For example, an application such as Firefox, which already runs on Windows on the x86 family, can be modified and re-built to run on Linux on the x86 (and potentially other architectures) as well. The multiple versions of the code may be stored as separate codebases, or merged into one codebase. An alternative to porting is cross-platform virtualization, where applications compiled for one platform can run on another without modification of the source code or binaries. As an example, Apple's Rosetta, which is built into Intel-based Macintosh computers, runs applications compiled for the previous generation of Macs that used PowerPC CPUs. Another example is IBM PowerVM Lx86, which allows Linux/x86 applications to run unmodified on the Linux/Power OS. Example of cross-platform binary software: The LibreOffice office suite is built for Microsoft Windows, macOS, Linux, FreeBSD, NetBSD, OpenBSD, Android, iOS, iPadOS, ChromeOS, web-based Collabora Online and many others.[8][9] Many of these are supported on several hardware platforms with processor architectures including IA-32, x86-64, ARM (ARMel, ARMhf, ARM64), MIPS, MIPSel, PowerPC, ppc64le, and S390x[9][10] A script can be considered to be cross-platform if its interpreter is available on multiple platforms and the script only uses the facilities built into the language. For example, a script written in Python for a Unix-like system will likely run with little or no modification on Windows, because Python also runs on Windows; indeed there are many implementations (e.g. IronPython for .NET Framework). The same goes for many of the open-source scripting languages. Unlike binary executable files, the same script can be used on all computers that have software to interpret the script. This is because the script is generally stored in plain text in a text file. There may be some trivial issues, such as the representation of a new line character. Some popular cross-platform scripting languages are: bash – A Unix shell commonly run on Linux and other modern Unix-like systems, as well as on Windows via the Cygwin POSIX compatibility layer, Git for Windows, or the Windows Subsystem for Linux. Perl – First released in 1987. Used for CGI programming, small system administration tasks, and more. PHP – Mostly used for web applications. Python – A language which focuses on rapid application development and ease of writing, instead of run-time efficiency. Ruby – An object-oriented language which aims to be easy to read. Can also be used on the web through Ruby on Rails. Tcl – A dynamic programming language, suitable for a wide range of uses, including web and desktop applications, networking, administration, testing and many more. Cross-platform or multi-platform is a term that can also apply to video games released on a range of video game consoles. Examples of cross-platform games include: Miner 2049er, Tomb Raider: Legend, FIFA series, NHL series and Minecraft. Each has been released across a variety of gaming platforms, such as the Wii, PlayStation 3, Xbox 360, personal computers, and mobile devices. Some platforms are harder to write for than others, requiring more time to develop the video game to the same standard. To offset this, a video game may be released on a few platforms first, then later on others. Typically, this happens when a new gaming system is released, because video game developers need to acquaint themselves with its hardware and software. Some games may not be cross-platform because of licensing agreements between developers and video game console manufacturers that limit development to one particular console. As an example, Disney could create a game with the intention of release on the latest Nintendo and Sony game consoles. Should Disney license the game with Sony first, it may be required to release the game solely on Sony's console for a short time or indefinitely. Main articles: Cross-platform play and List of video games that support cross-platform play Several developers have implemented ways to play games online while using different platforms. Psyonix, Epic Games, Microsoft, and Valve all possess technology that allows Xbox 360 and PlayStation 3 gamers to play with PC gamers, leaving the decision of which platform to use to consumers. The first game to allow this level of interactivity between PC and console games (Dreamcast with specially produced keyboard and mouse) was Quake 3.[11][12] Games that feature cross-platform online play include Rocket League, Final Fantasy XIV, Street Fighter V, Killer Instinct, Paragon and Fable Fortune, and Minecraft with its Better Together update on Windows 10, VR editions, Pocket Edition and Xbox One. Cross-platform programming is the practice of deliberately writing software to work on more than one platform. There are different ways to write a cross-platform application. One approach is to create multiple versions of the same software in different source trees—in other words, the Microsoft Windows version of an application might have one set of source code files and the Macintosh version another, while a FOSS *nix system might have a third. While this is straightforward, compared to developing for only one platform it can cost much more to pay a larger team or release products more slowly. It can also result in more bugs to be tracked and fixed. Another approach is to use software that hides the differences between the platforms. This abstraction layer insulates the application from the platform. Such applications are platform agnostic. Applications that run on the JVM are built this way. Some applications mix various methods of cross-platform programming to create the final application. An example is the Firefox web browser, which uses abstraction to build some of the lower-level components, with separate source subtrees for implementing platform-specific features (like the GUI), and the implementation of more than one scripting language to ease software portability. Firefox implements XUL, CSS and JavaScript for extending the browser, in addition to classic Netscape-style browser plugins. Much of the browser itself is written in XUL, CSS, and JavaScript. There are many tools[13][14] available to help the process of cross-platform programming: 8th: a development language which utilizes Juce as its GUI layer. It currently supports Android, iOS, Windows, macOS, Linux and Raspberry Pi. Anant Computing: A mobile application platform that works in all Indian languages, including their keyboards, and also supports AppWallet and native performance in all OSs. AppearIQ: a framework that supports the workflow of app development and deployment in an enterprise environment. Natively developed containers present hardware features of the mobile devices or tablets through an API to HTML5 code thus facilitating the development of mobile apps that run on different platforms. Boden: a UI framework written in C++. Cairo: a free software library used to provide a vector graphics-based, device-independent API. It is designed to provide primitives for 2-dimensional drawing across a number of different backends. Cairo is written in C and has bindings for many programming languages. Cocos2d: an open-source toolkit and game engine for developing 2D and simple 3D cross-platform games and applications. Codename One: an open-source Write Once Run Anywhere (WORA) framework for Java and Kotlin developers. Delphi: an IDE which uses a Pascal-based language for development. It supports Android, iOS, Windows, macOS, Linux. Ecere SDK: a GUI and 2D/3D graphics toolkit and IDE, written in eC and with support for additional languages such as C and Python. It supports Linux, FreeBSD, Windows, Android, macOS and the Web through Emscripten or Binaryen [Wikidata] (WebAssembly). Eclipse: an open-source development environment. Implemented in Java with a configurable architecture which supports many tools for software development. Add-ons are available for several languages, including Java and C++. FLTK: an open-source toolkit, but more lightweight because it restricts itself to the GUI. Flutter: A cross-platform UI framework for IOS, Android, Mac, Windows and developed by Google. fpGUI: An open-source widget toolkit that is completely implemented in Object Pascal. It currently supports Linux, Windows and a bit of Windows CE. GeneXus: A Windows rapid software development solution for cross-platform application creation and deployment based on knowledge representation and supporting C#, COBOL, Java including Android and BlackBerry smart devices, Objective-C for Apple mobile devices, RPG, Ruby, Visual Basic, and Visual FoxPro. GLBasic: A BASIC dialect and compiler that generates C++ code. It includes cross compilers for many platforms and supports numerous platform (Windows, Mac, Linux, Android, iOS and some exotic handhelds). Godot: an SDK which uses Godot Engine. GTK+: An open-source widget toolkit for Unix-like systems with X11 and Microsoft Windows. Haxe: An open-source language. Juce: An application framework written in C++, used to write native software on numerous systems (Microsoft Windows, POSIX, macOS), with no change to the code. Kivy: an open-source cross-platform UI framework written in Python. It supports Android, iOS, Linux, OS X, Windows and Raspberry Pi. LEADTOOLS: Cross-platform SDK libraries to integrate recognition, document, medical, imaging, and multimedia technologies into Windows, iOS, macOS, Android, Linux and web applications.[15] LiveCode: a commercial cross-platform rapid application development language inspired by HyperTalk. Lazarus: A programming environment for the FreePascal Compiler. It supports the creation of self-standing graphical and console applications and runs on Linux, MacOSX, iOS, Android, WinCE, Windows and WEB. Max/MSP: A visual programming language that encapsulates platform-independent code with a platform-specific runtime environment into applications for macOS and Windows A cross-platform Android runtime. It allows unmodified Android apps to run natively on iOS and macOS Mendix: a cloud-based low-code application development platform. MonoCross: an open-source model–view–controller design pattern where the model and controller are cross-platform but the view is platform-specific.[16] Mono: An open-source cross-platform version of Microsoft .NET (a framework for applications and programming languages) MoSync: an open-source SDK for mobile platform app development in the C++ family. Mozilla application framework: an open-source platform for building macOS, Windows and Linux applications. OpenGL: a 3D graphics library. Pixel Game Maker MV: A proprietary 2D game development software for Windows for developing Windows and Nintendo Switch games. PureBasic: a proprietary language and IDE for building macOS, Windows and Linux applications. ReNative: The universal development SDK to build multi-platform projects with React Native. Includes latest iOS, tvOS, Android, Android TV, Web, Tizen TV, Tizen Watch, LG webOS, macOS/OSX, Windows, KaiOS, Firefox OS and Firefox TV platforms. Qt: an application framework and widget toolkit for Unix-like systems with X11, Microsoft Windows, macOS, and other systems—available under both proprietary and open-source licenses. Simple and Fast Multimedia Library: A multimedia C++ API that provides low and high level access to graphics, input, audio, etc. Simple DirectMedia Layer: an open-source multimedia library written in C that creates an abstraction over various platforms' graphics, sound, and input APIs. It runs on OSs including Linux, Windows and macOS and is aimed at games and multimedia applications. Smartface: a native app development tool to create mobile applications for Android and iOS, using WYSIWYG design editor with JavaScript code editor. Tcl/Tk Titanium Mobile: open source cross-platform framework for Android and iOS development. U++: a C++ GUI framework for performance. It includes a set of libraries (GUI, SQL, etc..), and IDE. It supports Windows, macOS and Linux. Unity: Another cross-platform SDK which uses Unity Engine. Uno Platform: Windows, macOS, iOS, Android, WebAssembly and Linux using C#. Unreal: A cross-platform SDK which uses Unreal Engine. V-Play Engine: V-Play is a cross-platform development SDK based on the popular Qt framework. V-Play apps and games are created within Qt Creator. WaveMaker: A low-code development tool to create responsive web and hybrid mobile (Android & iOS) applications. WinDev: an Integrated Development Environment for Windows, Linux, .Net and Java, and web browers. Optimized for business and industrial applications. wxWidgets: an open-source widget toolkit that is also an application framework.[17] It runs on Unix-like systems with X11, Microsoft Windows and macOS. Xojo: a RAD IDE that uses an object-oriented programming language to compile desktop, web and iOS apps. Xojo supports natively compiling to Windows, macOS, iOS and Linux, and can also create compiled web apps that are able to be run as standalone servers or through CGI. This section possibly contains original research. Please improve it by verifying the claims made and adding inline citations. Statements consisting only of original research should be removed. (March 2025) (Learn how and when to remove this message) There are many challenges when developing cross-platform software: Testing cross-platform applications may be considerably more complicated, since different platforms can exhibit slightly different behaviors or subtle bugs. This problem has led some developers to deride cross-platform development as "write once, debug everywhere", a take on Sun Microsystems' "write once, run anywhere" marketing slogan. Developers are often restricted to using the lowest common denominator subset of features which are available on all platforms. This may hinder the application's performance or prohibit developers from using the most advanced features of each platform. Different platforms often have different user interface conventions, which cross-platform applications do not always accommodate. For example, applications developed for macOS and GNOME are supposed to place the most important button on the right-hand side of a window or dialog, whereas Microsoft Windows and KDE have the opposite convention. Though many of these differences are subtle, a cross-platform application which does not conform to these conventions may feel clunky or alien to the user. When working quickly, such opposing conventions may even result in data loss, such as in a dialog box confirming whether to save or discard changes. Scripting languages and VM bytecode must be translated into native executable code each time they are used, imposing a performance penalty. This penalty can be alleviated using techniques like just-in-time compilation; but some computational overhead may be unavoidable. Different platforms require the use of native package formats such as RPM and MSI. Multi-platform installers such as InstallAnywhere address this need. Cross-platform execution environments may suffer cross-platform security flaws, creating a fertile environment for cross-platform malware.[18] Operating context List of widget toolkits Hardware virtualization Language binding Source-to-source compiler Binary-code compatibility Comparison of user features of messaging platforms ^ "Design Guidelines: Glossary". java.sun.com. Archived from the original on 2012-02-13. Retrieved 2011-10-19. ^ "SDD Technology blog: Definition of cross platform". SDD Technology. Retrieved 2020-10-18. ^ Lee P Richardson (2016-02-16). "Xamarin vs Ionic: A Mobile, Cross Platform, Shootout". ^ a b "Platform Definition". The Linux Information Project. Retrieved 2014-03-27. ^ "About Mono". mono-project.com. Retrieved 2015-12-17. ^ Corti, Sascha P. (October 2011). "Browser and Feature Detection". MSDN Magazine. Retrieved 28 January 2014. ^ Choudhary, S.R. (2014). "Cross-platform testing and maintenance of web and mobile applications". Companion Proceedings of the 36th International Conference on Software Engineering. pp. 642–645. doi:10.1145/2591062.2591097. hdl:1853/53588. ISBN 9781450327688. S2CID 1903037. ^ Mehrotra, Pranob (2020-12-01). "Collabora Office suite gets a new layout for Android tablets and Chromebooks". XDA-Developers. Retrieved 2021-01-15. Collabora Office is a popular open-source alternative to the Microsoft Office suite. It's based on LibreOffice, and it's available on a variety of platforms, including Windows, Linux, iOS, and Android. This year in July, a major update for the office suite brought support for Chrome OS devices. ^ "Collabora Office on iOS and Android Just got Better!". Adfinis. 2020-12-15. Retrieved 2021-01-15. ...touch optimized interfaces: one for tablets and one for phone screens. ...(iOS, iPadOS, Chromebooks, Android). ^ "Nextcloud Ubuntu Appliance adds Collabora Online to Raspberry Pi image". MuyLinux. 2021-03-26. Retrieved 2021-03-30. the first viable self-hosted web office solution for the popular Raspberry Pi 4 platform ^ Cribba. Quake III Arena, Giant Bombcast, February 15, 2013. ^ A Closer Look At The Dreamcast Internet Starter Kit ^ The GUI Toolkit, Framework Page ^ "Platform Independent FAQ". Archived from the original on 2008-08-16. Retrieved 2009-04-25. ^ "Cross-Platform SDK Libraries for Recognition, Document, Medical, Imaging, and Multimedia". www.leadtools.com. Retrieved 2021-03-03. ^ "12 benefits of Xamarin Cross-platform app development". HeadWorks. 15 Mar 2019. ^ WxWidgets Description ^ Warren, Tom (2020-01-14). "Microsoft bids farewell to Windows 7 and the millions of PCs that still run it". The Verge. Retrieved 2020-02-06. Retrieved from " This tutorial will guide you through the process of creating a Jasper Report using Spring Boot.Jasper Reports is an open-source reporting library that allows developers to generate rich, dynamic, and customizable reports. It provides a powerful set of features for designing and rendering reports in various formats, such as PDF, HTML, Excel, etc. On the other hand, Spring Boot is a popular Java framework for building standalone, production-grade applications with ease.Our goal is to create a web application that generates a report using Jasper Reports. The application will utilize the Spring Boot framework to handle the backend operations and integrate Jasper Reports for report generation.We will start by setting up a new Spring Boot project and configuring the necessary dependencies. Then, we will design a report template using the JasperSoft Studio, a powerful visual report designer for Jasper Reports. Next, we will fetch data from a data source (e.g., a database) to populate the report. Once the data is retrieved, we will generate the report using the Jasper Reports API. Finally, we will display the generated report in a web application and explore options to export the report to different formats.Throughout the tutorial, we will provide code examples and explanations to help you understand each step clearly. By the end, you will have a working Spring Boot application that is creating and displaying Jasper Report.To begin, we need to set up a new Spring Boot project. You can use your preferred IDE or create the project from scratch using Maven or Gradle.Here's an example of a basic pom.xml file for a Maven-based Spring Boot project: org.springframework.boot spring-boot-starter-parent 2.5.2 org.springframework.boot spring-boot-starter-web Make sure to adjust the Spring Boot version according to your preference or project requirements.In this section, we will add the necessary dependencies to integrate Jasper Reports into our Spring Boot project.Jasper Reports provides a set of libraries and tools for report generation. To include these dependencies in our project, we need to update the pom.xml file with the following Maven dependencies: org.springframework.boot spring-boot-starter-web net.sf.jasperreports jasperreports 6.17.0 net.sf.jasperreports jasperreports-fonts 6.17.0 In the above example, we have added the Spring Boot Web Starter dependency as well as the Jasper Reports dependencies. The jasperreports artifact provides the core libraries for working with Jasper Reports, and the jasperreports-fonts artifact includes additional fonts that might be needed for rendering reports.Once you have added the dependencies, save the pom.xml file and let Maven download the necessary JAR files.Now that we have the dependencies set up, let's move on to designing the report template using JasperSoft Studio in the next section.In this section, we will design the report template using JasperSoft Studio. JasperSoft Studio is a visual report designer that allows us to create and customize the layout of our reports.To get started, you can download and install JasperSoft Studio from the official website. Once installed, follow these steps:Open JasperSoft Studio and create a new project.Right-click on the project and select "New Jasper Report."Provide a name for the report and choose the desired template (e.g., Blank A4).In the report editor, you can add elements such as text fields, images, tables, and charts to design the report layout.Customize the appearance of the elements by adjusting properties like font, color, size, etc.Bind the report elements to data fields by dragging and dropping them from the "Palette" onto the report.Save the report template.Once you have designed the report template, we need to import it into our Spring Boot project. Create a new directory src/main/resources/reports and place the report template file (with the .jrxml extension) inside it.In this section, we will fetch data from a data source to populate the report. For simplicity, let's assume we have a list of employees and we want to display their details in the report.First, create an Employee class representing the employee entity: public class Employee { private String firstName; private String lastName; private String designation; // Getters and setters } Next, create a service or repository class to fetch the employee data. For demonstration purposes, we will create a simple service class: @Service public class EmployeeService { public List getAllEmployees() { // Fetch employee data from a data source (e.g., database) // Return the list of employees } } Make sure to inject the EmployeeService wherever you need to retrieve the employee data.In this section, we will generate the Jasper Report using the report template and the fetched employee data.First, let's create a service class responsible for generating the report: @Service public class ReportService { private final EmployeeService employeeService; public ReportService(EmployeeService employeeService) { this.employeeService = employeeService; } public byte[] generateReport() throws Exception { List employees = employeeService.getAllEmployees(); // Load the report template InputStream reportTemplate = getClass().getResourceAsStream("/reports/employeeReport.jrxml"); // Compile the report template JasperReport jasperReport = JasperCompileManager.compileReport(reportTemplate); // Convert the list of employees to a JRBeanCollectionDataSource JRBeanCollectionDataSource dataSource = new JRBeanCollectionDataSource(employees); // Generate the report using the compiled template and data source JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport, null, dataSource); // Export the report to a byte array (PDF format) byte[] reportBytes = JasperExportManager.exportReportToPdf(jasperPrint); return reportBytes; } } In the above example, we inject the EmployeeService to fetch the employee data. We load the report template, compile it using JasperCompileManager, and convert the list of employees to a JRBeanCollectionDataSource. Finally, we generate the report using JasperFillManager and export it to a byte array (PDF format) using JasperExportManager.Note: In a real-world scenario, you may need to pass additional parameters to the report or apply more complex logic. The above code provides a basic example to get you started.In this section, we will create a Spring MVC controller to handle the report generation and display it in a web application.First, let's create the controller class: @Controller public class ReportController { private final ReportService reportService; public ReportController(ReportService reportService) { this.reportService = reportService; } @GetMapping("/report") public void generateReport(HttpServletResponse response) throws Exception { byte[] reportBytes = reportService.generateReport(); // Set the response headers response.setContentType("application/pdf"); response.setHeader("Content-Disposition", "inline; filename=employeeReport.pdf"); // Write the report bytes to the response output stream OutputStream outputStream = response.getOutputStream(); outputStream.write(reportBytes); outputStream.flush(); } }In the above example, we create a generateReport method mapped to the /report URL using the @GetMapping annotation. Inside the method, we obtain the generated report bytes from the ReportService and set the necessary response headers. Finally, we write the report bytes to the response output stream.Now, when you access the /report URL in your web application, it will generate and display the Jasper Report.In the next section, we will explore options to export the report to different formats.In this section, we will enhance our application to export the report to different formats, such as Excel and HTML.First, let's modify the ReportService to include methods for generating reports in different formats: @Service public class ReportService { // Existing code public byte[] generateReportAsExcel() throws Exception { List employees = employeeService.getAllEmployees(); InputStream reportTemplate = getClass().getResourceAsStream("/reports/employeeReport.jrxml"); JasperReport jasperReport = JasperCompileManager.compileReport(reportTemplate); JRBeanCollectionDataSource dataSource = new JRBeanCollectionDataSource(employees); JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport, null, dataSource); byte[] reportBytes = JasperExportManager.exportReportToPdf(jasperPrint); return reportBytes; } public byte[] generateReportAsHtml() throws Exception { List employees = employeeService.getAllEmployees(); InputStream reportTemplate = getClass().getResourceAsStream("/reports/employeeReport.jrxml"); JasperReport jasperReport = JasperCompileManager.compileReport(reportTemplate); JRBeanCollectionDataSource dataSource = new JRBeanCollectionDataSource(employees); JasperPrint jasperPrint = JasperFillManager.fillReport(jasperReport, null, dataSource); byte[] reportBytes = JasperExportManager.exportReportToPdf(jasperPrint); return reportBytes; } }In the above example, we added two new methods: generateReportAsExcel and generateReportAsHtml. These methods follow a similar pattern to the existing generateReport method, but export the report to Excel and HTML formats, respectively, using the JasperExportManager.exportReportToXlsx and JasperExportManager.exportReportToHtml methods.Next, modify the ReportController to handle the export functionality: @Controller public class ReportController { // Existing code @GetMapping("/report/excel") public void generateReportAsExcel(HttpServletResponse response) throws Exception { byte[] reportBytes = reportService.generateReportAsExcel(); response.setContentType("application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"); response.setHeader("Content-Disposition", "attachment; filename=employeeReport.xlsx"); OutputStream outputStream = response.getOutputStream(); outputStream.write(reportBytes); outputStream.flush(); } @GetMapping("/report/html") public void generateReportAsHtml(HttpServletResponse response) throws Exception { byte[] reportBytes = reportService.generateReportAsHtml(); response.setContentType("text/html"); response.setHeader("Content-Disposition", "attachment; filename=employeeReport.html"); OutputStream outputStream = response.getOutputStream(); outputStream.write(reportBytes); outputStream.flush(); } }In the above example, we added two new methods in the ReportController: generateReportAsExcel and generateReportAsHtml. These methods handle the export of the report to Excel and HTML formats, respectively. We set the appropriate response headers based on the desired format and write the report bytes to the response output stream.Now, you can access the /report/excel and /report/html URLs in your web application to download the report in Excel and HTML formats, respectively.In this tutorial, we have learned about creating a Jasper Report with Spring Boot. We started by setting up a Spring Boot project and adding the necessary dependencies for Jasper Reports. Then, we designed a report template using JasperSoft Studio and fetched data from a data source to populate the report.By following the steps outlined in this tutorial, you should now have a good understanding of how to create Jasper Reports with Spring Boot and leverage its powerful reporting capabilities in your applications. Feel free to experiment and enhance the application further based on your specific requirements.