Vysokoškolská skripta

VŠE

FAKULTA INFORMATIKY A STATISTIKY

NÁSTROJE PRO TVORBU, SPRÁVU A NASAZOVÁNÍ APLIKACÍ

Ing. Filip Vencovský, Ph.D.

Tato publikace byla vytvořena za podpory projektu Interní rozvojové soutěže Vysoké školy ekonomické v Praze s názvem Příprava prostředí a výukových materiálů pro podporu předmětu Softwarové inženýrství (IRS/F4/20/2019).

Oponentem skript je doc. Ing. Alena Buchalcevová, Ph.D.

Pokud není uvedeno jinak, jsou autory obrázků ve skriptech autoři skript.

© Vysoká škola ekonomická v Praze, Nakladatelství Oeconomica – Praha 2019
ISBN 978-80-245-2341-5

Obsah

Seznam obrázků

Kapitola 1

Úvod

Elektronická skripta jsou určena studentům kurzu **4IT115 Softwarové inženýrství** a vyžadují základní znalost programovacího jazyka *Java*.

Skripta přináší informace o nástrojích, technologiích a doporučených postupech pro vývoj desktopových aplikací v jazyce *Java* s pomocí grafické knihovny *JavaFX*.

Skripta jsou především prakticky orientovaná. Provádí čtenáře od vytvoření softwarového projektu pro knihovnu *JavaFX* až po doručení výsledného produktu a vybavují ho vším, co potřebuje vědět pro efektivní týmovou práci.

Skripta nemají za cíl replikovat dokumentaci použitých technologií a nástrojů, nepřináší ani žádné nové poznatky a inovativní postupy. Za cíl naopak mají **přinést čtenáři podstatné informace a umožnit mu orientovat se v technologiích, doporučených postupech a nejčastějších problémech**. Odkazují se proto na další zdroje, které souvisí s prezentovanými tématy, např. na dokumentaci, odborné články, blogové příspěvky, hesla na Wikipedii, odborné knihy dostupné online, výuková videa na YouTube a názory komunity na webu.

Skripta se zaměřují na tato témata:

- založení nového projektu (kapitola 7),
- sdílení existujícího projektu na vzdálený repozitář (kapitola 8),
- klonování a úprava existujícího projektu ze vzdáleného repozitáře (kapitola 9),
- volba architektury JavaFX aplikace (kapitola 10),
- sestavení projektu (kapitola 11),
- týmová práce na projektu (kapitola 12),
- vydání hotové aplikace (kapitola 13).

Použité postupy demonstrují vzájemné propojení těchto technologií:

- grafická knihovna *JavaFX* (kapitola 3),
- vývojové prostředí *IntelliJ IDEA* (kapitola 4),
- nástroj na správu verzí *Git* (kapitola 5),
- nástroj na správu a sestavení projektu *Maven* (kapitola 6),
- vzdálený repozitář a nástroj na podporu týmové práce a automatizaci *GitLab*.

Skripta jsou primárně určená pro čtení na elektronických zařízeních. Obsahují hypertextové odkazy, které obsah rozšiřují a umožňují hlouběji proniknout do studované problematiky. V tištěné formě není možné odkazy použít, z tohoto důvodu není doporučené skripta tisknout.

Živá verze skript je hostovaná na gitlab.com/FIS-VSE/studijni-materialy/softwarove-inzenyrstvi.

Kapitola 2

Java

Java označuje softwarovou platformu a objektově orientovaný programovací jazyk. Java umožňuje spouštět stejný kód na různých operačních systémech a zařízeních pomocí mezivrstvy virtuálního stroje (Java Virtual Machine).

Kapitola vám pomůže porozumět rozdílu ve verzích, edicích a odlišných licenčních podmínkách implementací Javy. Najdete tu také odkaz na instalační návody a binární soubory Javy.

Pokud byste váhali, zda je Java pro vás to pravé:

- Podle hackr.io patří mezi několik **nejlepších voleb** pro budoucí uplatnění kvůli snadnému učení a skvělým pracovním příležitostem.
- V žebříčku popularity PYPL je na **druhém místě**, za *Pythonem* a před *JavaScriptem*.
- V průzkumu StackOverflow je na **pátém místě**, po *JavaScriptu*, *HTML*, *SQL* a *Pythonu*. Podle stejného průzkumu nepatří u vývojářů mezi nejmilovanější nebo nejžádanější, ale ani mezi nejstrašnější.
- Na **třetím místě**, po *JavaScriptu* a *Pythonu*, se podle GitHut drží v objemu aktivit na GitHubu.

Všechny údaje jsou z léta 2019. Podrobnosti o dalších jazycích najdete např. v článku na Codinginfinite.

2.1 Oracle JDK a OpenJDK

Javu primárně vyvíjí společnost Oracle pod označením *Oracle JDK*.

Oracle JDK s sebou nese **licenční omezení** a umožňuje pouze vývoj, prototypování a vlastní užití. Za produkční nasazení se odvádí poplatky. Podrobné čtení nabízí plný text licence.

Alternativně lze použít svobodnou variantu implementace Javy s otevřeným kódem v podobě projektu ***OpenJDK***.

Implementace OpenJDK je také podporovaná společností Oracle, ale vyvíjí ji komunita pod licencí **GPLv2 s linkovací výjimkou**. Výjimku obsahuje proto, aby bylo možné použít pro svůj program Javu a zároveň pro svůj kód použít libovolnou licenci.

Odkazy na další informace o licenčních podmínkách:

- plný text licence OpenJDK,
- čtení o linkovací výjimce,
- čtení o GPL,
- situaci okolo licencí Javy skvěle mapuje komunitou vyvíjený dokument Java Is Still Free.

2.2 Verze

Podobně jako například Linux rozlišuje Java mezi stabilními verzemi s dlouhou podporou (*LTS*) a standardními edicemi. Více informací v článku Long-term support na Wikipedii.

V době vydání skript jsou k dispozici dvě LTS verze – 8 a 11.

Podle průzkumu mezi vývojáři (2018 a 2019) byla pro produkční prostředí stále preferovaná **verze 8**. Jelikož byla verze 11 vydána v září 2018, dá se předpokládat, že bude její využití teprve pomalu narůstat.

Rozdíly mezi jednotlivými verzemi jsou dobře popsány v článku Java version history na Wikipedii a do verze 11 shrnuty v dotazu Summary of differences between Java versions? na Stackexchange.

2.3 Instalace

Javu lze instalovat buď jako prostředí pro běh programů psaných v Javě (dříve JRE, nyní bez označení), nebo spolu s kompilátorem a nástroji pro vývoj (JDK). V rámci vývoje je tedy logické, že budeme používat JDK.

Stažení a instalace JDK

- Oracle JDK je ke stažení na oracle.com.
- OpenJDK: *návod pro Linux najdete na openjdk.java.net,* binární soubory OpenJDK jsou ke stažení na adoptopenjdk.net.

Instalaci je možné ověřit příkazem `java -version`.

Ujistěte se, že máte nastavenou proměnnou `JAVA_HOME` a že odkazuje na požadovanou instalaci Javy. Návod na její nastavení je například v článku Set JAVA_HOME on Windows 7, 8, 10, Mac OS X, Linux.

Pokud máte nainstalováno více verzí Javy, je možné mezi nimi přepínat. Na Linuxu například pomocí příkazu `update-alternatives --config java`.

2.4 Další edice

Text kapitoly i celá skripta se věnují **Standard Edition (SE)**. Jak již název napovídá, jedná se o standardní edici Javy.

Enterprise Edition (EE) je komunitou vyvíjená edice usnadňující tvorbu komplexních podnikových aplikací s důrazem na technologie a principy jako HTML5, REST a WebSocket. Edice je spojená zejména s aplikačním serverem GlassFish. Více informací najdete v knize *Your First Cup: An Introduction to the Java EE Platform*.

2.5 Tutoriály

Oficiální tutoriály k použití grafiky, zvuku a k práci s časovými údaji nebo připojení k databázi najdete na docs.oracle.com/javase/tutorial/.

Kapitola 3

JavaFX

JavaFX, nebo také *OpenJFX*, je open source grafická knihovna pro **desktopové aplikace** psané v Javě. Za vývojem stojí komunita a podporu zajišťuje společnost Gluon. Stejně jako OpenJDK je knihovna pod licencí **GPLv2 s linkovací výjimkou**. Ačkoliv knihovnu lze použít i pro vývoj mobilních aplikací, není to v praxi obvyklé.

Oficiální stránku knihovny najdete na openjfx.io.

Kapitola vám pomůže orientovat se v možnostech přípravy knihovny k použití ve vývoji aplikace a jiných existujících knihovnách pro vývoj desktopových aplikací na platformě Java.

3.1 Příprava JavaFX k použití

V Oracle JDK 8 je JavaFX ještě součástí standardního balíčku. Od verze 11 a také v případě OpenJDK je třeba JavaFX **nainstalovat zvlášť**.

Při instalaci je třeba věnovat pozornost **verzi knihovny JavaFX**. Ta by měla být shodná s verzí Javy, kterou se rozhodnete používat pro projekt.

3.1.1 Instalace na disk

Binární soubory příslušné verze JavaFX SDK je možné stáhnout na gluonhq.com/products/javafx.

Po stažení a rozbalení archivu je třeba odkázat na složku s JavaFX proměnnou `PATH_TO_FX`. Ta se nastaví obdobně jako proměnná `JAVA_HOME` popsaná v předchozí kapitole. Návod na instalaci najdete na oficiálních stránkách openjfx.io/openjfx-docs.

Pokud z nějakého důvodu potřebujete pracovat s OpenJDK 8, musíte se poohlédnout po adekvátní verzi JavaFX 8:

- Ubuntu 16.04 LTS disponuje ve standardním balíčku *openjfx*, v 18.04 LTS také, ale s vynucením verze `sudo apt install *openjfx*=8*`.
- V případě Windows jsou binární soubory dostupné na GitHub.com v projektu openjfx-win.
- Na Mac OS nejsou žádné binární soubory k dispozici. Jedinou možností je sestavit verzi 8 ze zdrojového kódu.

3.1.2 Použití závislostí (Maven)

Druhou variantou pro používání knihovny JavaFX je provázání pomocí *Maven* závislostí. V případě připojení knihovny jako závislosti není třeba nic předem instalovat.

Použití závislostí je doporučené jen pro projekty s **Java SDK 11 a vyššími verzemi**, protože v Maven repozitáři nejsou k dispozici binární soubory pro verzi 8.

Nástroji Maven se budeme věnovat ve zvláštní kapitole **Správa projektu**. Použití JavaFX v Maven projektu je součástí kapitoly **Založení FX projektu**.

Návod na nastavení projektu s JavaFX pomocí Maven najdete také na oficiálních stránkách openjfx.io/openjfx-docs.

3.2 Alternativní grafické knihovny

Ačkoli v minulosti existovala řada podobných knihoven, JavaFX nemá pro desktopové aplikace na Java platformě konkurenci. Seznam alternativních knihoven najdete například v článku Java applications.

Na internetu se nezřídka objevují srovnání s knihovnami Swing nebo AWT, které jsou předchůdci JavaFX. Ty jsou sice stále součástí Javy 8 a 11, ale v dalších verzích už nebudou podporovány. Příklad srovnání najdete v článku JavaFX vs Swing.

3.3 Další zdroje

Dodatečné informace ke knihovně JavaFX si můžete přečíst v dokumentaci JavaFX SDK 8 nebo na původní stránce OpenJFX.

Kapitola 4

Vývojové prostředí

Kapitola se věnuje výhodám použití integrovaného vývojového prostředí, doporučeným nástrojům a odkazům na stažení těchto nástrojů.

Vývojové prostředí neboli IDE (*Integrated Development Environment*) je navrženo pro zvýšení efektivity práce vývojáře. Život vám usnadní zejména následující funkce:

- autocomplete (doplňování jmen proměnných, metod, tříd, cest),
- upozorňování na chyby v kódu (a často i návrhy na opravu),
- snadná navigace v projektu a souborech,
- generování kódu (konstruktory, get a set metody),
- refaktoring (bezpečné přejmenování tříd, metod a proměnných),
- snadný přechod na deklaraci (metody, proměnné, třídy),
- snadný přechod nebo zobrazení dokumentace,
- generování dokumentace,
- generování class diagramu,
- integrace debuggeru (procházení programu),
- správa nastavení projektu,
- automatické sestavení projektu,
- integrace správy verzí,
- spouštění testů.

Můžete se setkat i s názory, že namísto IDE je lepší použít lehčí editor (*Vim*, *VisualStudio Code*, *Atom*) v kombinaci s dalšími nástroji nebo pluginy na sestavení a testování projektu. Pro vývoj v Javě to ale není příliš obvyklé.

Odpovědi z reálného světa můžete prozkoumat na StackOverflow.

4.1 Doporučená prostředí

Pro vývoj v Javě jsou v praxi nejpoužívanější tři nástroje:

- IntelliJ IDEA (jetbrains.com/idea),
- Apache NetBeans (netbeans.apache.org),
- Eclipse IDE for Java Developers (www.eclipse.org).

Všechna zmíněná prostředí jsou k dispozici pro Windows, Mac i Linux.

Ačkoli podle **preferencí uživatelů** (Slant) je jejich pořadí stejné jako ve výše uvedeném seznamu, vývojáři na svá oblíbená IDE nedají dopustit. Pokud se chcete lépe zorientovat, skvěle poslouží článek Martina Hellera.

Seznam dalších možných IDE pro Javu najdete v článku na hackr.io.

4.2 IntelliJ IDEA

IntelliJ IDEA je k dispozici v komunitní edici *Community* nebo profesionální edici *Ultimate*. Ačkoli je pro projekty v rámci předmětu Softwarové inženýrství dostačující komunitní verze, je možné si požádat o studentskou licenci profesionální verze na jetbrains.com/student.

Pokud si chcete přečíst o rozdílech ve funkcionalitě více, JetBrains připravilo podrobné srovnání edic.

Edice *Community* je k dispozici pod licencí Apache 2.0 a umožňuje i použití pro komerční projekty. Studentská licence pro *Ultimate* komerční využití neumožňuje.

Nástroj IntelliJ IDEA **stáhnete** na jetbrains.com/idea/download.

Pro Linux je také k dispozici jako Snap (*Community*, *Ultimate*) nebo Flatpack (*Community*, *Ultimate*).

Kapitola 5

Správa verzí

Na správu verzí lze obecně nahlížet jako na činnost vedoucí ke sledování a organizaci postupných změn na určitých artefaktech.

V knize Pro Git se o správě verzí hovoří jako o systému, který zaznamenává změny souboru nebo sady souborů v čase tak, abyste se mohli později k určité verzi vrátit.

Kapitola se zaměřuje na systém pro správu verzí *Git*, který je de facto standardem pro správu verzí při vývoji softwaru.

5.1 Git

Git je open source systém pro **správu verzí** pod licencí GPLv2.

Oficiální stránky projektu najdete na git-scm.com.

Hlavní charakteristiky systému Git:

- Patří mezi **distribuované systémy**. Na rozdíl od centralizovaných systémů (SVN nebo CVS) si drží celý obraz úložiště a jeho historii nejen na serveru, ale také u klientů.
- Umožňuje pracovat s **více než jedním** vzdáleným úložištěm.
- Podporuje **nelineární vývoj**. Můžete tak pracovat ve více různých větvích a následně je spojovat.
- Nefunguje na bázi seznamu přírůstků, ale udržuje si **celé obrazy úložiště** v každé verzi.
- Kontroluje integritu dat pomocí hashování.

Výhody jsou shrnuty na oficiálních stránkách v sekci About.

Ačkoli stále existují projekty používající Subversion (SVN) nebo Mercurial, Git patří v současnosti mezi nejoblíbenější nástroje. Na **srovnání** se můžete podívat např. na Stackshare či Quora a na statistiky open source projektů na OpenHub. V roce 2016 dokonce vznikla stránka https://svnvsgit.com/, která se snaží bořit mýty, které vznikly z nadšení pro nový systém.

Pro seznámení s Gitem a hlubší porozumění je nejlepší **kniha Pro Git** a její tři první kapitoly.

Za zhlédnutí stojí **výuková videa** ze série Git Basics o verzovacích systémech (5:58), systému Git (8:15) a práci s Gitem (4:27).

5.2 Instalace Gitu

Pro Windows a Mac je možné **stáhnout** Git z oficiálních stránek v sekci Downloads.

Na Linuxu bývá součástí systémových repozitářů pod názvem `Git`.

S Gitem lze pracovat několika způsoby:

- přímo z příkazové řádky,
- přímo z IDE,
- pomocí externího nástroje.

Některá IDE mají v sobě Git integrovaný a není jej třeba instalovat. V případě **IntelliJ IDEA** musíte odkázat na binární soubor Gitu na disku, abyste jej mohli v IDE používat.

Pokud jste na Windows a nemáte právo instalace, můžete na oficiálních stránkách stáhnout přenosnou verzi (*Portable*).

Seznam externích klientů najdete např. v článku na Hostinger. V rámci předmětu je doporučené jako externí nástroj používat **GitKraken**, který funguje na operačních systémech Windows, Mac i Linux a nabízí dobrý uživatelský zážitek.

5.3 Vzdálené repozitáře

Ačkoli je to možné, nemělo by příliš velký smysl Git používat bez vzdáleného úložiště *(repozitář = úložiště)*, obvykle označovaného jako *origin*. Buď je možné jej hostovat na vlastním serveru, nebo použít jako službu. Projekty s otevřeným zdrojovým kódem je většinou možné hostovat na serverech zdarma.

Mezi **nejoblíbenější služby** patří:

- GitLab,
- GitHub
- a Bitbucket.

Srovnání dle preferencí uživatelů můžete najít na Slant.

Zmíněné služby kromě prostoru pro verzovaný kód nabízí další **užitečné funkce** pro podporu vývoje jako zejména sledování problémů (*issue tracking*), revize kódu (*code revisions*), řízení žádostí o spojování větví (*merge requests*), správa dokumentace, integrace do dalších nástrojů, řízení přístupu k větvím a celkové řízení přístupových práv nad úložištěm.

GitLab jako open source platforma získal na oblíbenosti mezi vývojáři hlavně kvůli zabudování mechanismů kontinuální integrace a nasazení (*Continual Integration* a *Continual Deployment*, zkratka ***CI/CD***). Po odeslání kódu do úložiště se automaticky provede otestování, sestavení projektu a nasazení. Jako vlastníci projektu pak máte jistotu, že sestavení patří ke konkrétní verzi a prošlo automatizovanými testy.

5.4 Práce se systémem Git

Git pracuje se soubory v adresáři jako obvykle. Rozdíl je v tom, že Git v adresáři sleduje, co se změnilo od poslední verze (revize).

Když se rozhodnete změny zapsat jako novou revizi, přidáte změněné soubory podle svého uvážení do **seznamu změn** (*stage*). Seznam změn následně zapíšete jako novou revizi s dobře zvoleným komentářem, abyste se vy i další vývojáři v úložišti dobře vyznali.

Ve skutečnosti se soubory mohou nacházet ve **čtyřech různých stavech**: *untracked*, *modified*, *staged*, *unmodified*/*commited*. Podrobnosti najdete v kapitole Nahrávání změn do repozitáře.

Mezi základní funkce Gitu patří:

- zaznamenání změn jako nové revize (`commit`),
- přidání souboru do seznamu změn (`add`),
- informace o stavu úložiště (`status`),
- zobrazení změn oproti aktuálnímu stavu nebo jiné revizi (`diff`),
- inicializace prázdného repozitáře (`init`),
- klonování vzdáleného úložiště (`clone`),
- vytvoření nové větve (`branch`),

- přepnutí na jinou větev nebo revizi (`checkout`),
- uchování neodeslaných změn při přepínání mezi revizemi a větvemi (`stash`).

Podrobné vysvětlení funkcí najdete v kapitole 2 knihy Pro Git. Pro rychlou orientaci poslouží tahák.

Na YouTube uvidíte nekomentovaný příklad práce s Gitem v příkazové řádce (10:18) a v GitKraken (14:07), který s komentářem naplno zažijete na přednášce.

Tutoriál pro práci v **NetBeans** najdete v článku Using Git Support in NetBeans IDE. Práce s Gitem v **IntelliJ IDEA** je demonstrovaná na praktické ukázce v dalších kapitolách.

5.5 Git workflow

Klíčovým faktorem úspěšnosti při použití Gitu, zejména při spolupráci týmu vývojářů, je stanovení způsobu práce s větvemi (*Git workflow*).

V principu je doporučené chránit **hlavní větev (*master*)** proti zápisu běžných změn a ponechat ji jen pro větší fungující celky. Běžné změny se potom odehrávají ve **vývojové větvi**, označované obvykle jako *dev* nebo *develop*. Je na zvážení každého týmu, kdy se uzavře onen větší fungující celek a dojde k připojení vývojové větve na hlavní.

Poměrně běžné je vytváření dalších větví při vývoji nové funkce, opravy chyby nebo ladění hotové verze programu.

Strategie, které je možné zvolit, jsou názorně popsané v **tutoriálu** nebo článku Feature branching your way to greatness od Atlassian.

Kapitola 6

Správa projektu

Správu projektu tvoří aktivity, které vedou ke snadné údržbě kódu, správě závislostí, výstupních artefaktů a plánu spouštění testů a sestavení aplikace.

O správu a sestavení projektu se dokáže postarat IDE. Nastavení projektu a potřebné knihovny navolíte v konfiguračních dialogových oknech. Pokud ale nechcete, aby byl projekt závislý na konkrétním vývojovém prostředí a jeho verzi, je vhodné použít jeden ze systémů správy a sestavení projektu. Zejména pokud projekt koncipujete jako veřejně dostupný s otevřeným zdrojovým kódem, stává se to de facto standardem. Další důvod pro použití systému na správu a sestavení projektu je kontinuální integrace a nasazení (*CI/CD*), které z nastavení systému mohou čerpat.

Oblíbené nástroje, které se pro tento problém hodí, jsou **Maven** a **Gradle**. Jejich porovnání, včetně staršího nástroje pro sestavení *Ant*, shrnuje článek na Baeldung. Na Slant není zatím mnoho hlasujících, ale už tam můžete vidět v bodech základní výhody a nevýhody obou nástrojů.

6.1 Gradle

Gradle je široce použitelný nástroj pro automatizaci, který stejně jako Maven řeší i konfiguraci projektu, závislosti a sestavení. Pro konfiguraci používá jazyk Groovy. Veškeré funkce se zadávají pomocí pluginů a jejich konfigurace.

6.2 Maven

Maven je nástroj na správu projektu vyvíjený organizací *Apache*. Oficiální stránku projektu s dokumentací najdete na maven.apache.org. Až se budete chtít blíže seznámit s Maven, je vhodné pročíst si Getting Started Guide.

Pokud dodržíte předepsané konvence, Maven dokáže vyřešit většinu problémů. V opačném případě jich dokáže naopak spoustu přinést. Konvence například předepisují, jaká má být adresářová struktura projektu, kde se mají hledat zdroje a jak probíhá sestavení.

Pokud by jeho použití mělo mít jen jednu praktickou výhodu, bude to **správa závislostí**. Při vývoji aplikace se nevyhnete použití dalšího kódu. Kód se obvykle stane součástí projektu připojením externích knihoven. Je ale náročné spravovat různé verze externích knihoven a propojovat je vždy s projektem v každém novém prostředí. Maven je napojený na rozsáhlý repozitář MvnRepository, ve kterém najdete velké množství projektů s uspořádanými vydáními knihoven. Po zapsání krátké deklarace o závislosti na dalších projektech (*dependency*) se kód automaticky stáhne do projektu a můžete ho hned použít.

Konfigurace projektu se tvoří na jednom místě v souboru `pom.xml`, který je zkratkou anglického názvu objektového modelu projektu (*Project Object Model*). Jak napovídá přípona, konfigurace se tvoří v jazyce XML. Konvence předepisuje, že soubor by se měl nacházet v kořeni projektu.

O práci na sestavení projektu se starají **pluginy**. Pluginy jsou už předkonfigurované, ale jejich nastavení je možné překrýt vlastním. Seznam a dokumentaci nejběžnějších pluginů najdete na maven.apache.org/plugins. Například plugin `jar` slouží k zabalení programu do spustitelného archivu. Mimo to můžete použít i pluginy třetích stran, například plugin na sestavení JavaFX projektu od OpenJFX.

Nástrojem, který zefektivňuje práci na vytvoření projektu, jsou předdefinované šablony (**archetypy**). S pomocí archetypu vytvoří Maven strukturu projektových složek a předdefinovaný objektový model. Archetypy mohou obsahovat i ukázkové soubory. Například *javafx-archetype-simple* lze použít pro vytvoření jednoduchého JavaFX projektu. Podrobně se s archetypy můžete seznámit v Úvodu do archetypů v oficiální dokumentaci.

6.2.1 Identifikace Maven projektu

Každý Java projekt má třídy organizované do balíčků. Maven projekty dodržují konvenci pro názvy balíčků a jejich hierarchickou strukturu tak, aby **neexistovaly** žádné dvě třídy se stejným jménem a stejnou strukturou balíčků (*classpath*).

To se hodí zejména k tomu, aby bylo možné použít v projektu jakoukoli jinou existující knihovnu a třídu, aniž by došlo ke konfliktu. Je to pochopitelně také jedna z podmínek pro uveřejnění projektu v oficiálním Maven repozitáři.

Organizace tříd do balíčků začíná v případě Maven projektu už u identifikačních údajů projektu. Když zakládáte nový projekt, musíte vyplnit *GroupId* a *ArtifactId*. Více najdete v dokumentaci projektu v části o jmenných konvencích.

Vezměme v úvahu existující příklad systému CHVote pro pořádání voleb ve Švýcarsku, který je veřejně dostupný na github.com/republique-et-canton-de-geneve/chvote-1-0. V projektu na GitHub.com je k dispozici i jeho objektový model (POM).

GroupId by měl představovat unikátní název projektu. Drží se konvencí Javy pro unikátní názvy balíčků. Není tedy možné používat speciální znaky a názvy musí dodržovat *velbloudí notaci*. *GroupId* je tvořen doménou, kterou organizace vlastní. Pořadí je ale opačné oproti zvyklostem na webu. První je doména prvního řádu, následuje druhý a případně třetí řád. V uvedeném příkladu software produkuje kanton Ženeve s webovou adresou *www.ge.ch*. *GroupId* proto bude začínat `ch.ge`. Následuje hierarchická struktura projektů a modulů, které vlastník pod doménu řadí. V uvedeném příkladu je název projektu *ve* (*vote électronique*). *GroupId* je tedy `ch.ge.ve`.

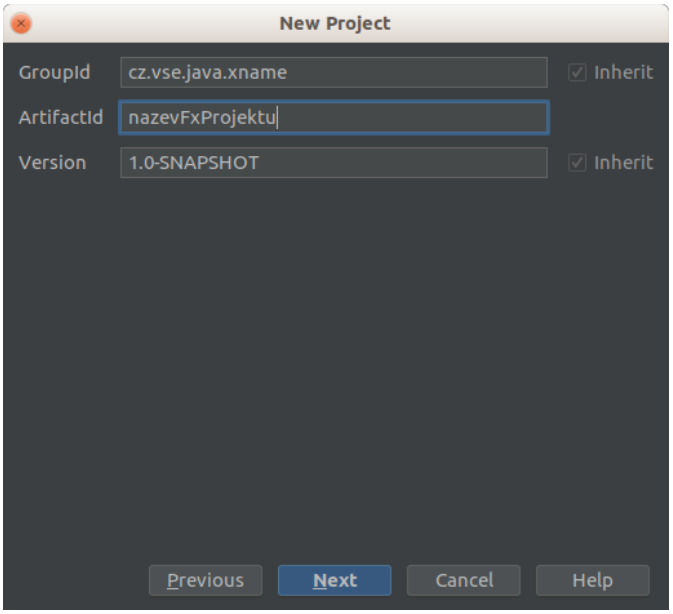
ArtifactId je název produktu (*artefaktu*) v rámci projektu. Pod tímto názvem se potom bude produkt distribuovat. V příkladu `ch.ge.ve` jsou názvy artefaktu `base-pom`, `admin-offline`, `commons-base`.

Version je posledním údajem, který je třeba vyplnit a označuje verzi artefaktu. Pro verzování existují různé zvyklosti. Doporučeným způsobem je sémantické verzování, které se skládá ze tří číslic ve formátu `MAJOR.MINOR.PATCH`. Podrobnosti o sémantickém verzování najdete na semver.org.

Součástí názvu verze bývá typicky příznak ***SNAPSHOT***, např. `1.0.0-SNAPSHOT`, který označuje, že verze je stále ve vývoji a mění se na každodenní bázi. Finální produkty už nesmí být označeny tímto příznakem. Pro více informací si přečtěte vysvětlení komunity na StackOverflow nebo What is a SNAPHOT version? ze stránek Maven.

Jiným příkladem z open source světa je legislativní databáze státu New York na github.com/nysenate/OpenLegislation s objektovým modelem (POM), ve kterém najdete *GroupId* `gov.nysenate` a *ArtifactId* `legislation`.

Pro projekt v rámci předmětu Softwarové inženýrství použijeme **jednotnou formu** ***GroupId*** `cz.vse.java.xname`, kde *xname* je váš login do školního systému. *ArtifactId* vyplňte podle požadavků na konkrétní projekt v rámci cvičení.



Obrázek 6.1: Identifikace nového projektu

6.2.2 Project Object Model (POM)

Projektový model je místo, ve kterém je uložena konfigurace Maven projektu. Tvoří ho XML soubor s názvem `pom.xml`.

Všechny Maven projekty jsou **potomkem** tzv. Super POM.

Definice objektového modelu **musí vždy zahrnovat** kořenový element `project`. V něm by měla být verze objektového modelu (element *modelVersion*), ze kterého bude vycházet ten váš. V současnosti se používá verze `4.0.0`. Dále je nutné uvést výše zmíněné identifikační údaje `GroupId`, `ArtifactId` a `version` (viz Minimal POM).

```
<project>
  <modelVersion>4.0.0</modelVersion>
  <GroupId>cz.vse.java.xname</GroupId>
  <ArtifactId>nazevFxProjektu</ArtifactId>
  <version>1.0-SNAPSHOT</version>
</project>
```

Kromě identifikačních údajů je vhodné uvést také další informace o projektu jako `description`, `url`, `licences`, `organization`, `scm`, `issueManagement`, `contributors`, `developers`. Pro bližší informace o všech elementech, které jsou součástí POM, si prohlédněte dokumentaci Maven.

Závislosti na dalších projektech se tvoří pomocí elementu `dependencies`, do kterého jsou vložené jednotlivé `dependency` s identifikačními údaji závislosti. Závislosti se během sestavování vyhledají a stáhnou z centrálního repozitáře. Další informace o závislostech jsou v průvodci How to use external dependencies a dokumentaci.

```
<project>

    <!-- jiné části POM -->

    <dependencies>
        <dependency>
            <GroupId>org.junit.jupiter</GroupId>
            <ArtifactId>junit-jupiter-engine</ArtifactId>
            <version>5.4.2</version>
            <scope>test</scope>
        </dependency>
    </dependencies>

    <!-- jiné části POM -->

</project>
```

6.2.3 Struktura souborů Maven projektu

Výchozí konfigurace Maven projektu předpokládá, že veškeré soubory projektu budou umístěné v určité struktuře složek. Více informací najdete na webu Maven v části Introduction to the standard directory layout.

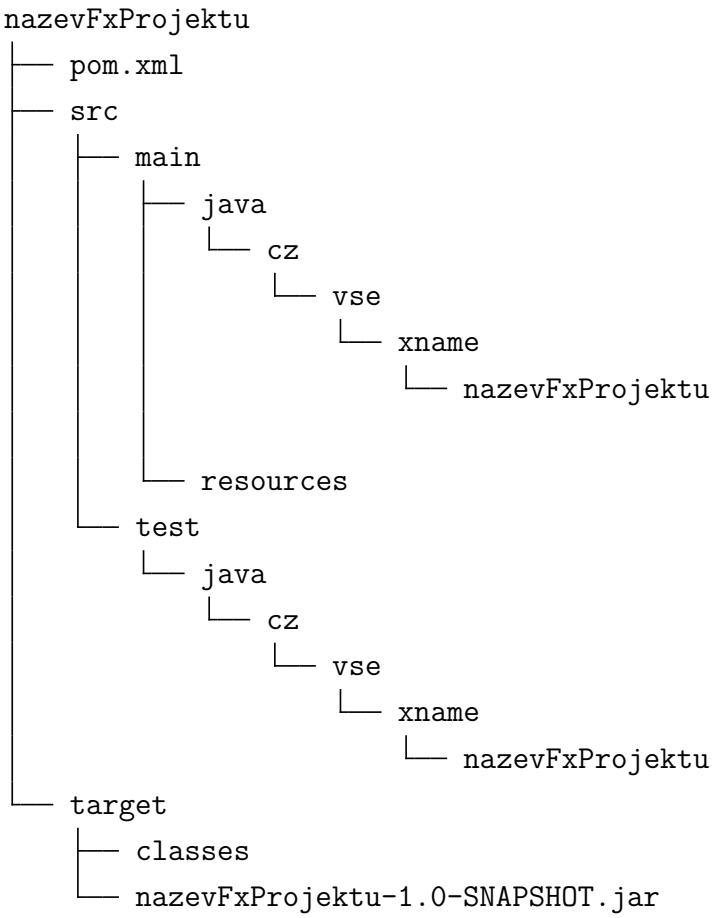
Objektový model se nachází v kořeni projektového adresáře.

Zdrojový kód se nachází v kořeni projektového adresáře ve složce `src`. Složka se dále dělí na `main` a `test`. Obě složky uvnitř dodržují strukturu, která odpovídá názvu balíčku a identifikačním údajům projektu.

Pokud Váš projekt používá **zdroje** jako například obrázky, zvukové soubory nebo HTML, Maven je bude očekávat ve složce `src/main/resources`.

Složku `class` se **zkompilovanými třídami** najdete v kořeni adresáře ve složce `target`, což je odlišné od jiných zvyklostí, kde narazíte na složky jako *build* nebo *out*.

Sestavený archiv najdete taktéž v kořeni adresáře ve složce `target`.



Výše uvedené schéma demonstruje strukturu složek typického Maven projektu.

Kapitola 7

Nový JavaFX projekt

Kapitola popisuje postup vytvoření nového projektu založeného na grafické knihovně JavaFX a vývojovém prostředí IntelliJ IDEA.

7.1 Varianty založení nového JavaFX projektu

Nový JavaFX projekt lze vytvořit několika způsoby:

- varianta A: jako Maven projekt s Oracle JDK 8,
- varianta B: jako Maven projekt z JavaFX archetypu s JDK 11,
- varianta C: jako Maven projekt s OpenJDK 8 a ručním napojením na JavaFX,
- varianta D: jako Maven projekt s JDK 11 a ručním napojením na JavaFX,
- varianta E: jako JavaFX projekt.

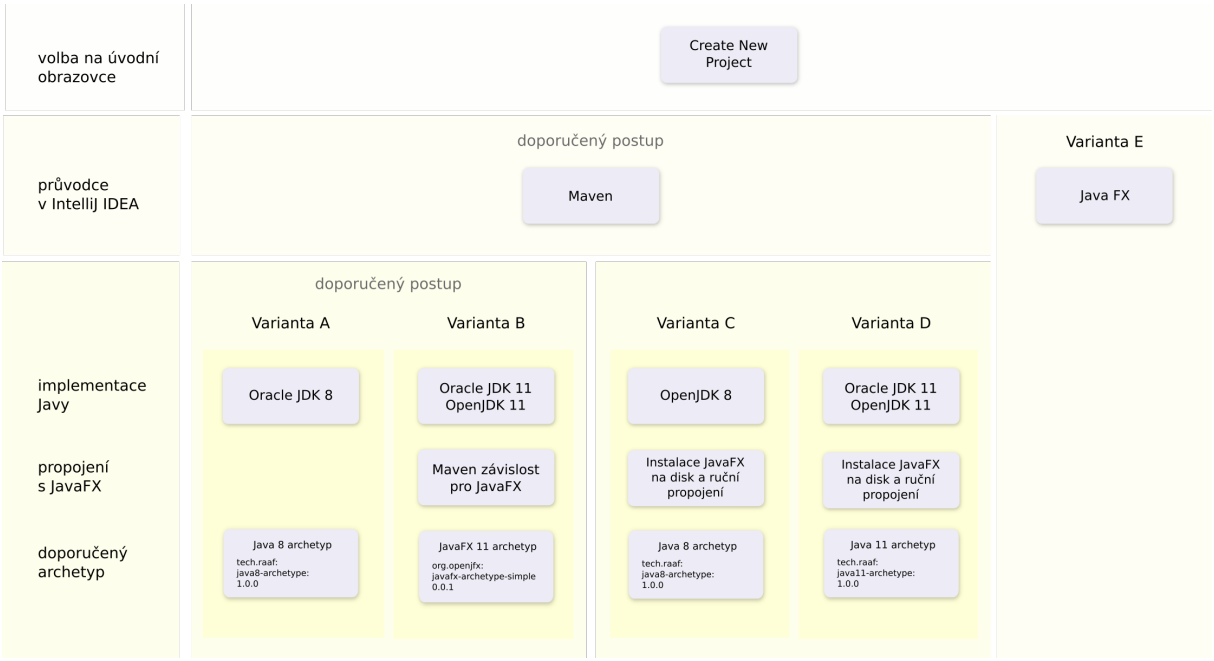
Při vytváření projektu **z archetypu JavaFX** (varianta B) je výhodou, že se nemusíte starat o instalaci JavaFX (viz kapitola JavaFX), která se sama stáhne z Maven repozitáře a bude ihned k použití v projektu. Z důvodu nedostupných repozitářů pro JavaFX verze 8 je tento postup možný **jen pro Javu 11** a vyšší verze.

Při vytváření JavaFX projektu **pomocí průvodce** (varianta E) sice projekt vytvoříte, ale pokud nemáte Javu ve verzi 8 na Windows (tj. máte vyšší verzi nebo jiný OS), budete muset do projektu ručně přidat archiv knihovny JavaFX (stejně jako u variant C a D). Navíc bude projekt obtížné sestavovat na počítačích s jiným IDE z důvodu chybějící konfigurace.

Ve všech ostatních případech (varianty A, C a D) se nevytvoří ukázkové soubory JavaFX, ale s knihovnou bude možné pracovat, protože v projektu bude existovat jako Maven

závislost. Stejně tak můžete knihovnu JavaFX jako závislost použít v každém již existujícím projektu.

Pro lepší pochopení možností při zakládání nového JavaFX projektu si prohlédněte následující schéma:



Obrázek 7.1: Schéma zakládání projektu

Na stránkách nápovědy k IntelliJ IDEA najdete také návod na vytvoření Maven projektu (hodí se pro varianty A, B, C, D).

7.2 Maven projekt s Oracle JDK 8 – varianta A

Podmínky pro uplatnění postupu:

- Je nainstalovaná **Java Oracle JDK verze 8**.

Připojení knihovny JavaFX:

- Jelikož je JavaFX součástí Oracle JDK verze 8, postup záměrně **neřeší její připojení** do projektu.

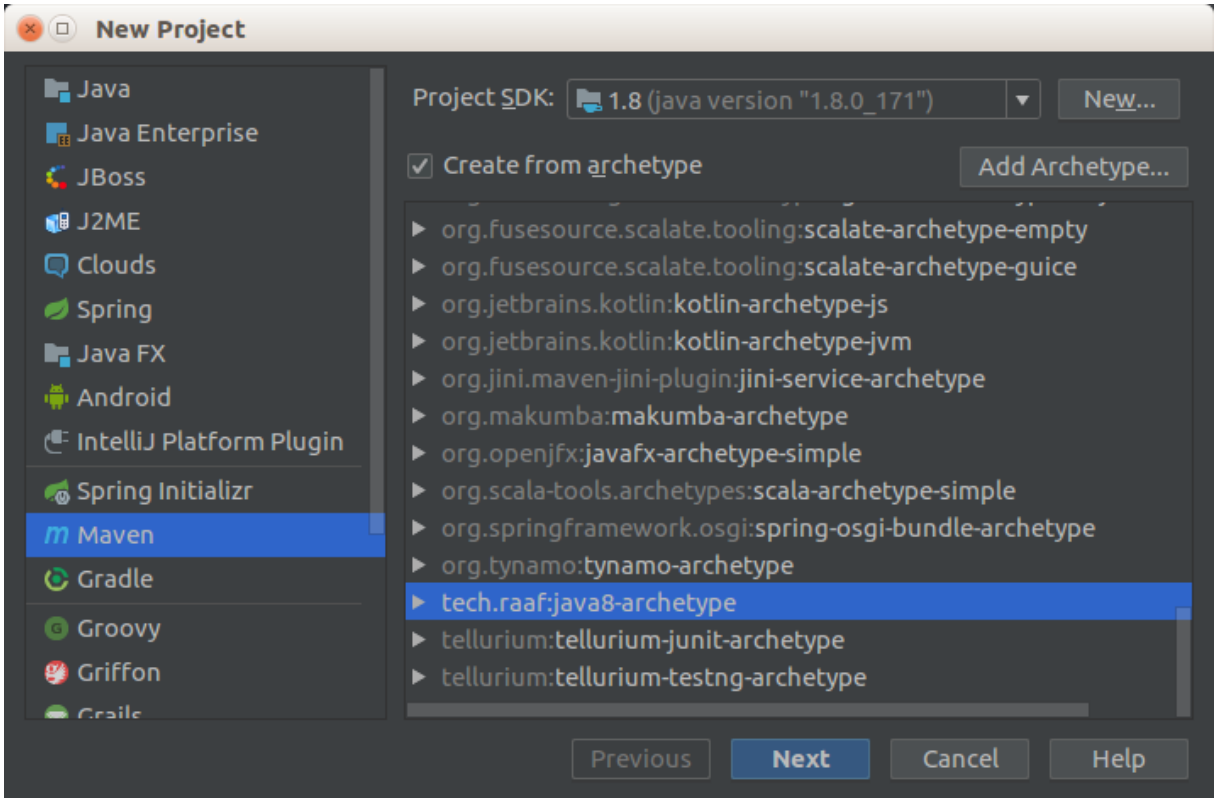
Jedná se o doporučený postup. V případě, že vám podmínky neumožňují vytvořit projekt tímto způsobem, prozkoumejte další varianty.

7.2.1 Postup založení projektu

1. Po spuštění IntelliJ IDEA zvolit na úvodní obrazovce *Create New Project*.

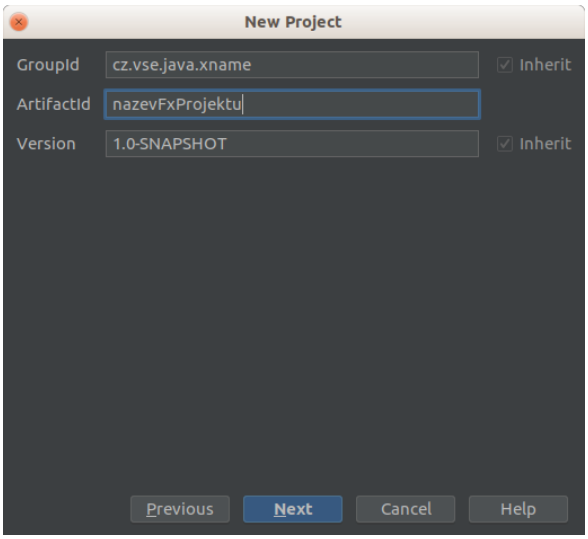
Obrázek 7.2: Úvodní obrazovka IntelliJ

2. V levém menu vybrat *Maven*.
3. Z rozbalovacího seznamu vybrat SDK 1.8.
4. Zaškrtnout volbu *Create from archetype*.
5. Vybrat archetyp `tech.raaf:java8-archetype`. Pokud archetyp neexistuje, přidejte ho jako nový archetyp.



Obrázek 7.3: Vybrat java8-archetype

6. Zvolit SDK Javy, který chcete používat. V tomto postupu SDK 8, označený jako *1.8*.
7. Potvrdit SDK a archetyp tlačítkem *Next*.
8. Vyplnit identifikační údaje nového Maven projektu (viz kapitola **Správa projektu > Identifikace Maven projektu**).
9. Potvrdit identifikační údaje tlačítkem *Next*.

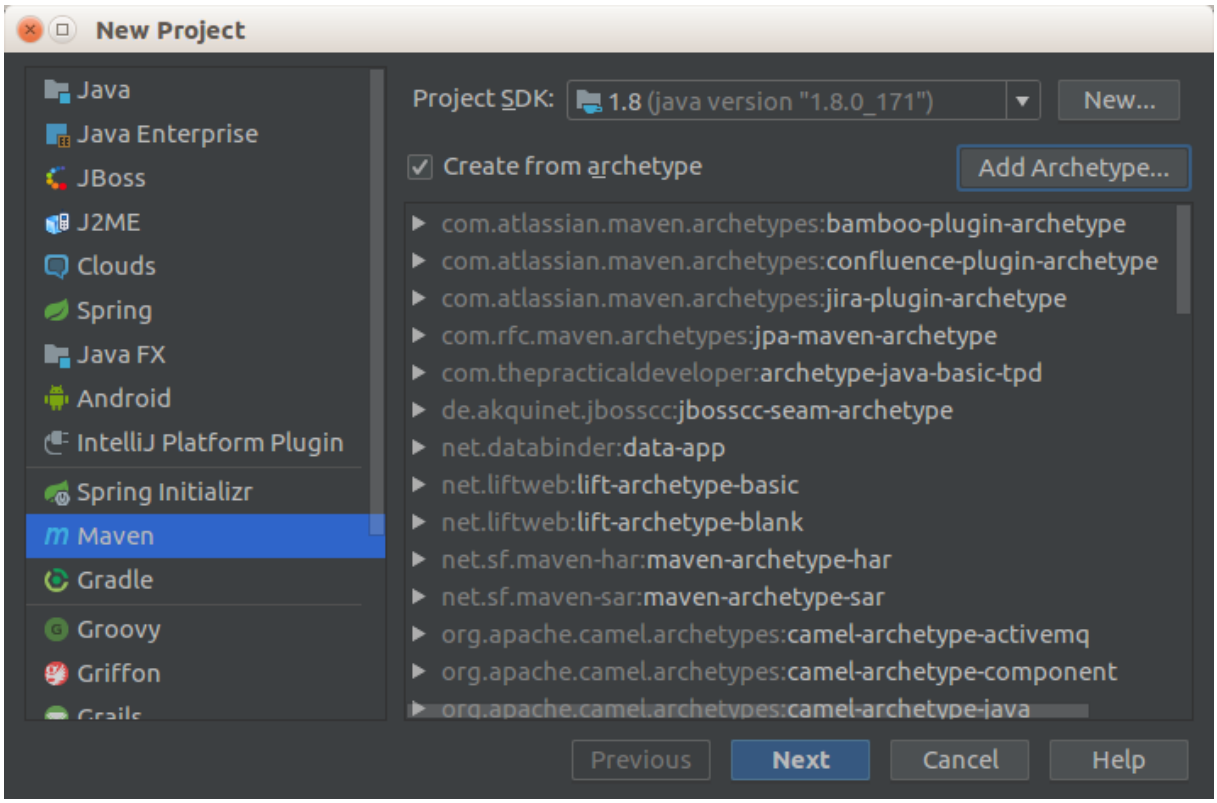


Obrázek 7.4: Identifikace nového projektu

7.2.2 Přidání archetypu, pokud neexistuje

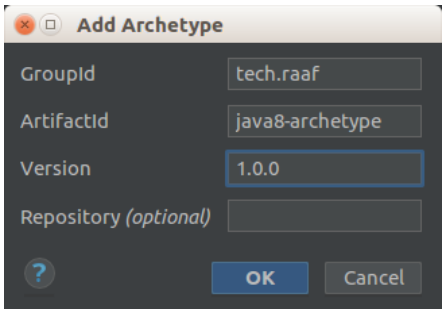
Pokud se v bodě 4 v seznamu archetypů nenacházel `tech.raaf:java8-archetype`, je třeba ho přidat jako **nový archetyp** následujícím způsobem:

4a. Vpravo nahoře kliknout na tlačítko *Add Archetype...*



Obrázek 7.5: Přidat archetyp

4b. Vyplnit údaje archetypu `tech.raaf:java8-archetype` dle Maven repozitáře.

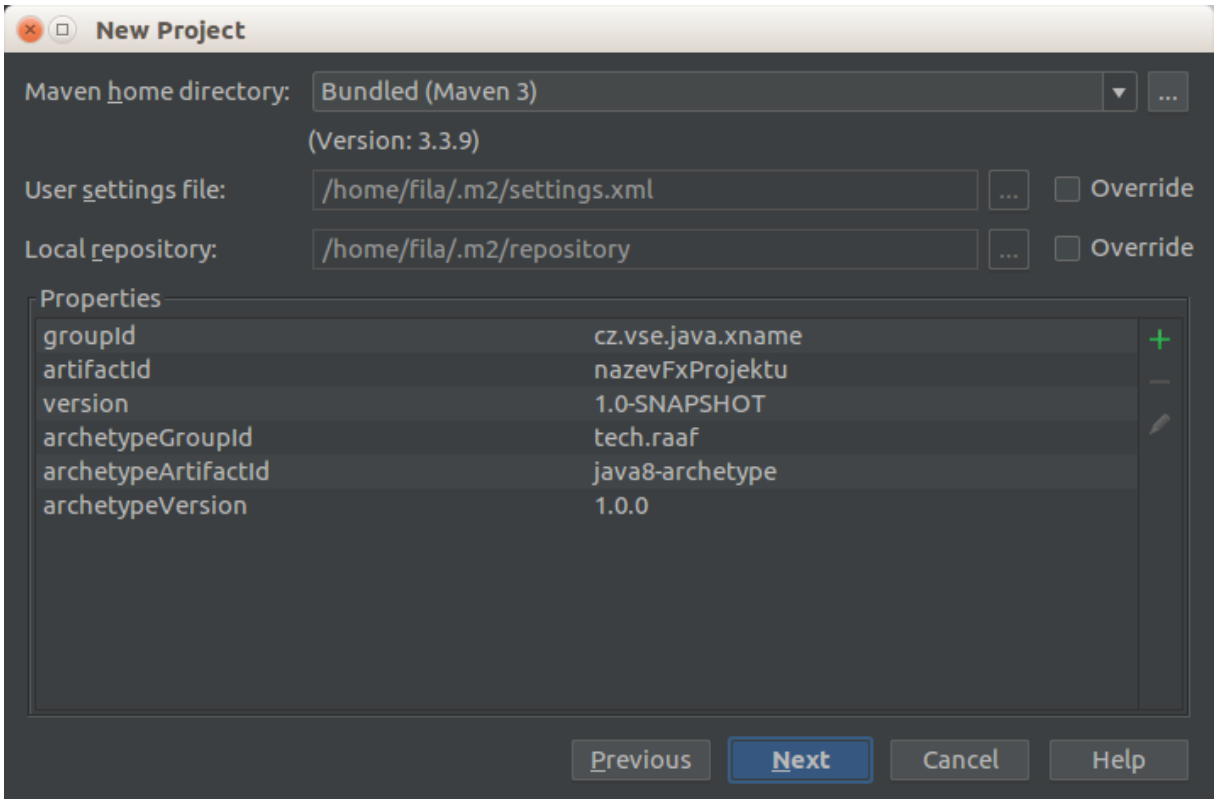


Obrázek 7.6: Přidat nový archetyp Java 8

7.2.3 Dokončení postupu

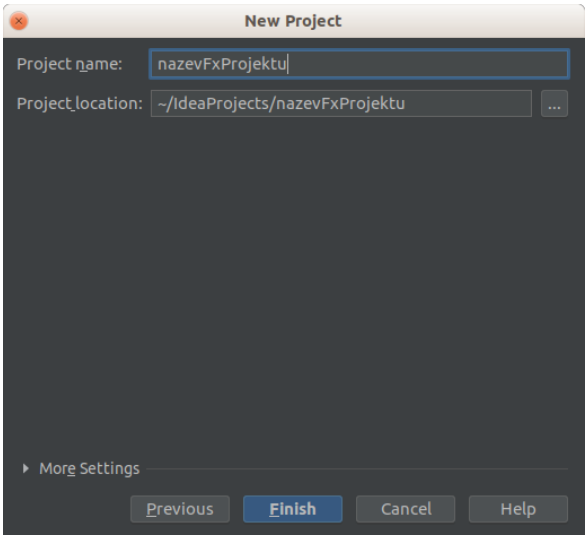
Po potvrzení identifikačních údajů nového Maven projektu (groupId, artifactId, version):

10. Potvrďte výchozí nastavení pro použití Maven (soubor s nastavením a umístění staženého kódu) a vlastností projektu tlačítkem *Next*.



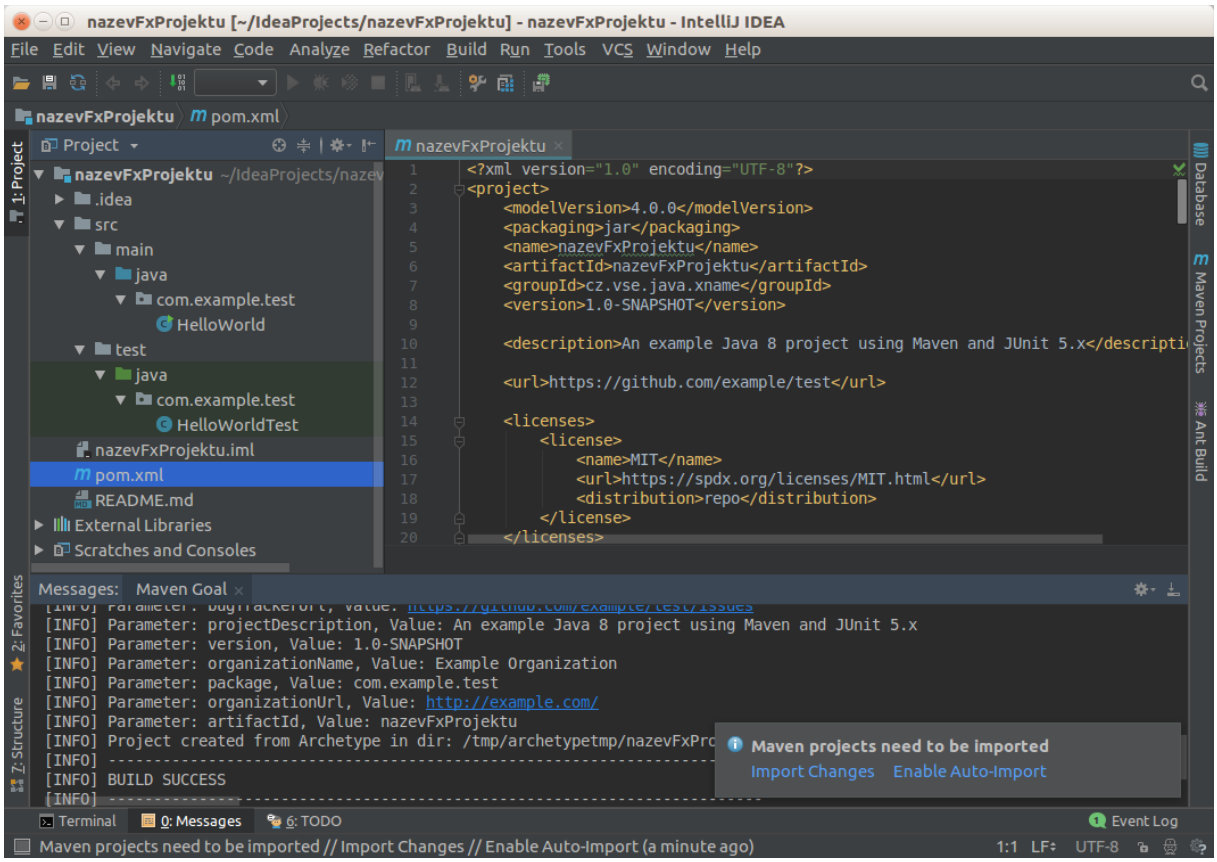
Obrázek 7.7: Vlastnosti nového projektu

11. V poslední obrazovce dialogu potvrďte název projektu a jeho umístění na disku tlačítkem *Finish*.



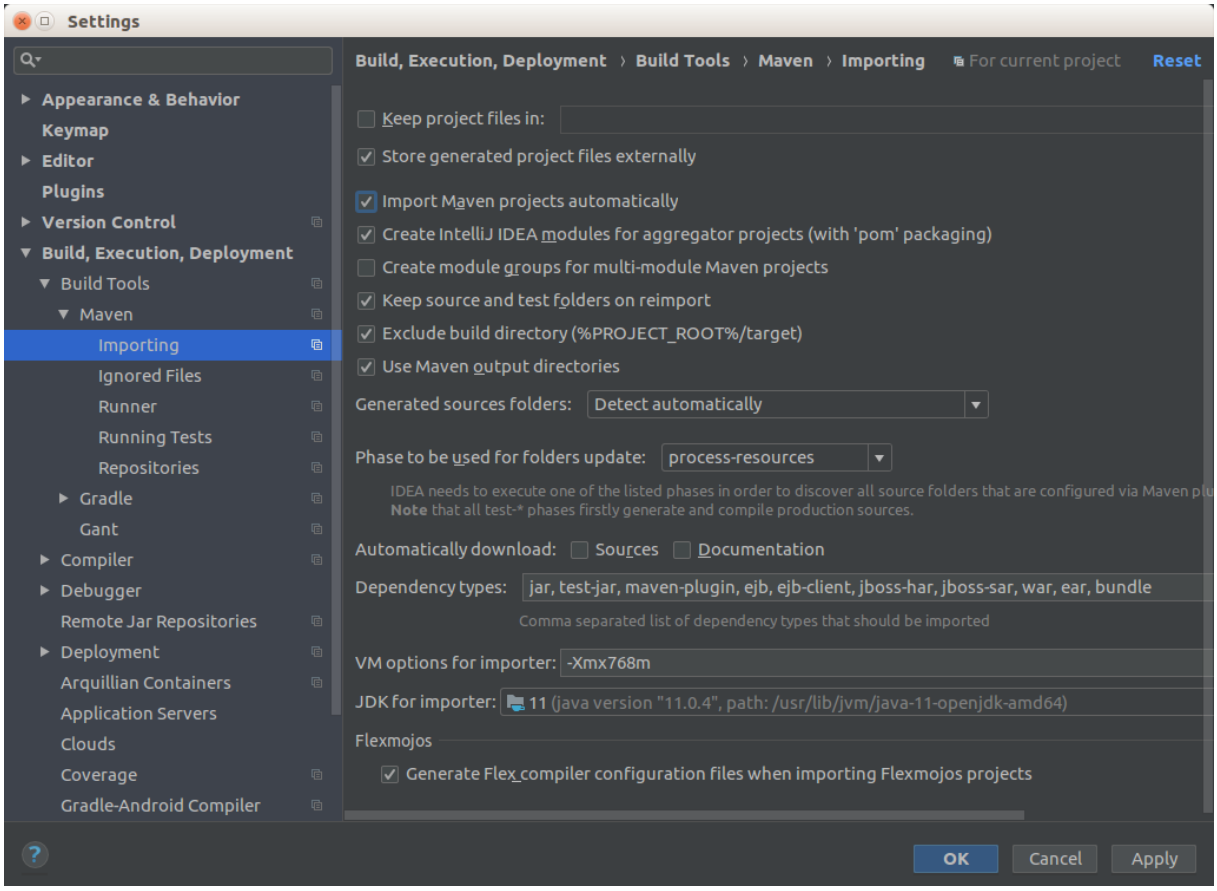
Obrázek 7.8: Název a umístění projektu

Projekt se vytvořil a zobrazila se struktura souborů a nově vytvořený objektový model (POM).



Obrázek 7.9: Vytvořený projekt

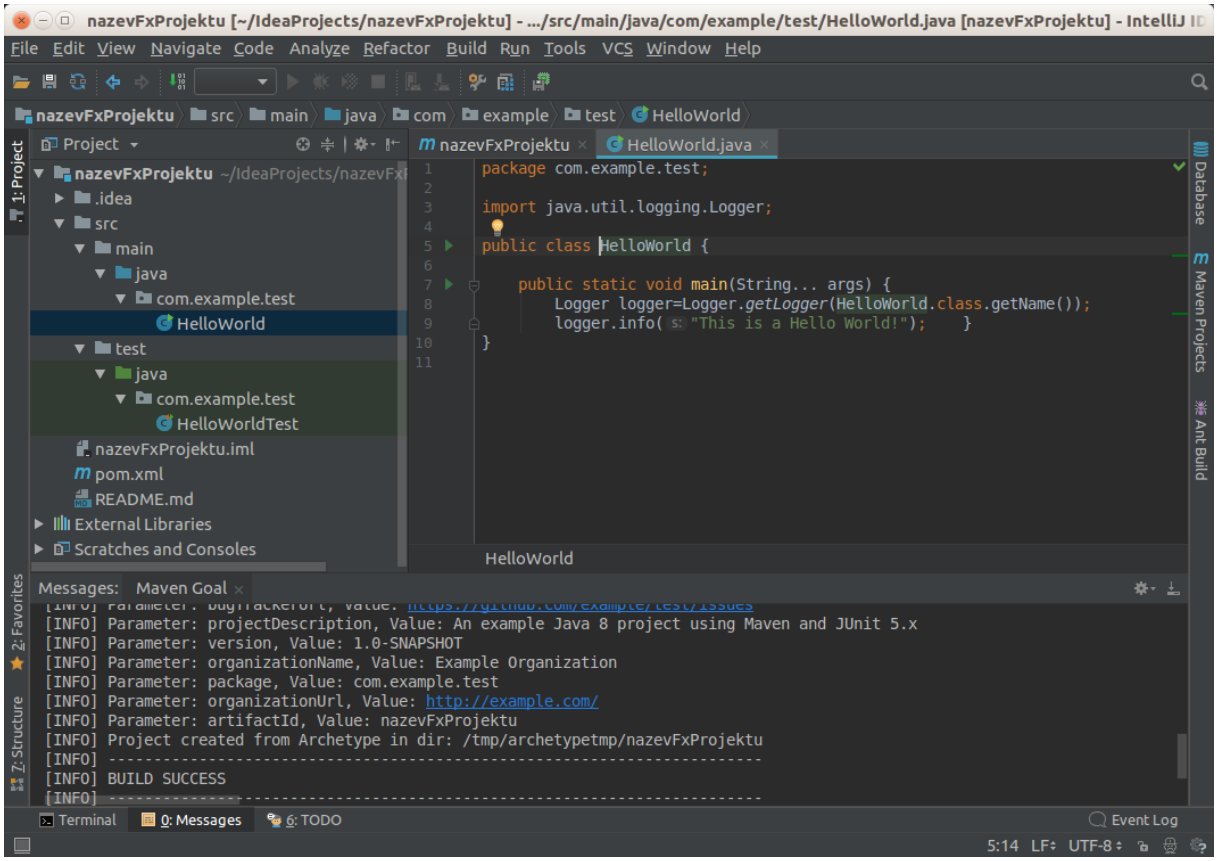
12. Pozornost věnujte malému informačnímu dialogu vpravo dole, kde je možné spravovat vlastnosti **importu**. Doporučené je kliknout na volbu *Enable Auto-Import*, která zaručí, že knihovny se budou stahovat automaticky při změně objektového modelu. Vlastnosti importu můžete upravovat i později v nastavení v dialogu *Build, Execution, Deployment > Build Tools > Maven > Importing* zaškrtnutím volby *Import Maven projects automatically*.



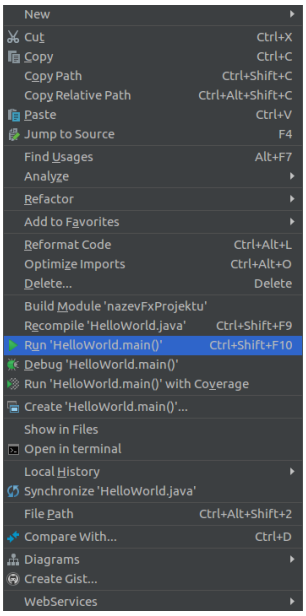
Obrázek 7.10: Nastavení importu

7.2.4 Spuštění projektu

Projekt se spustí kliknutím na spustitelnou třídu (s metodou *main*) pravým tlačítkem a vybráním volby *Run*.



Obrázek 7.11: Spustitelná třída



Obrázek 7.12: Kontextové menu třídy

Alternativně je možné vybrat volbu *Run* z kategorie *Run* na horní liště. V některých případech IDEA nenavrhne třídu ke spuštění a je třeba ji poprvé spustit z kontextové

nabídky. Vytvoří se tím konfigurace spuštění. Jinak je možné konfiguraci spuštění přidat také ručně volbou *Run...* v kategorii *Run* a následně vybrat v seznamu na pravé straně *Application*.

Po spuštění se objeví v konzoli ve spodní části obrazovky text *"Hello World"*.

Jak je zřejmé, aplikace zatím nepoužívá JavaFX knihovnu a netvoří žádné okno.

7.3 Maven projekt s JDK 11 – varianta B

Podmínky pro uplatnění postupu:

- Je nainstalovaná Java Oracle JDK 11 a OpenJDK 11.

Připojení knihovny JavaFX:

- Jelikož JavaFX není součástí JDK 11, je do projektu připojena jako Maven závislost.

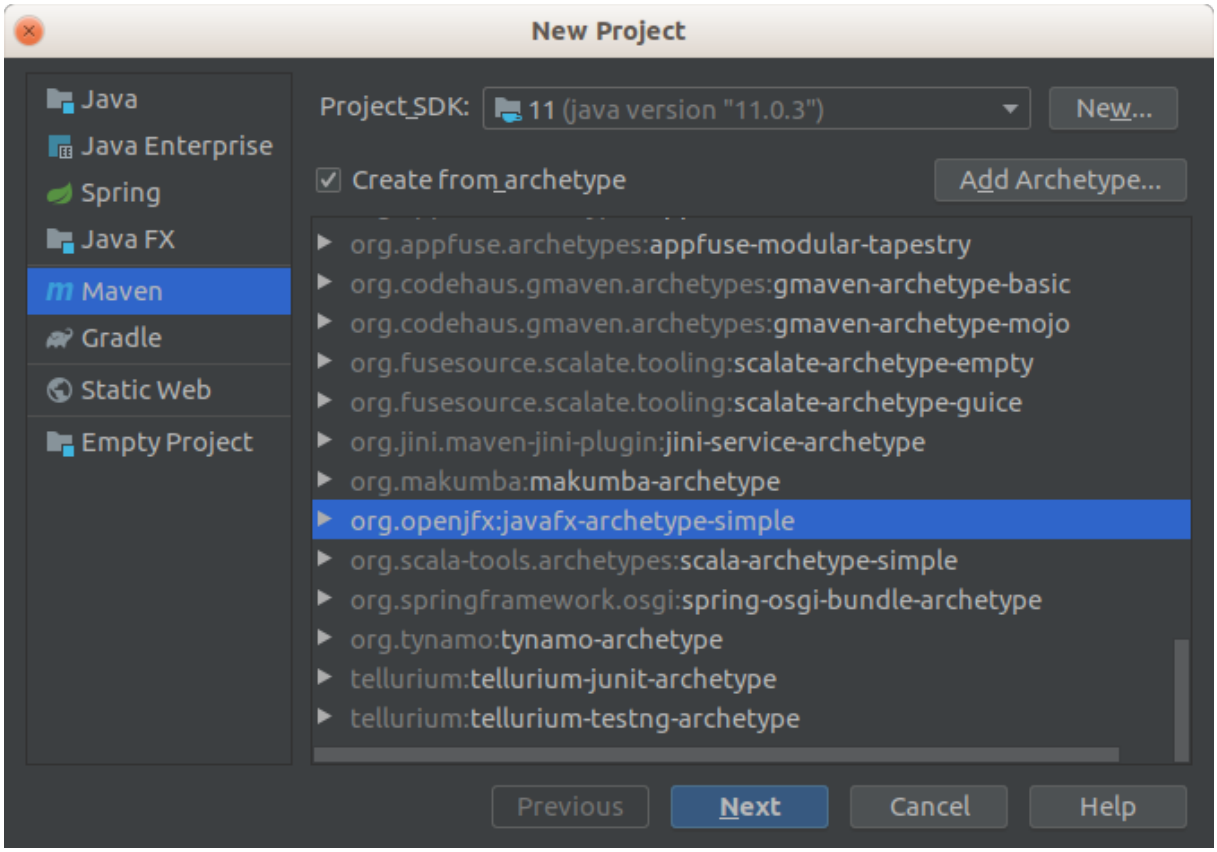
Jedná se o doporučený postup v případě, že není možné nainstalovat Java Oracle JDK verze 8. Pokud vám podmínky neumožňují vytvořit projekt tímto způsobem, prozkoumejte další varianty.

7.3.1 Postup založení projektu

1. Po spuštění IntelliJ IDEA zvolit na úvodní obrazovce *Create New Project*.

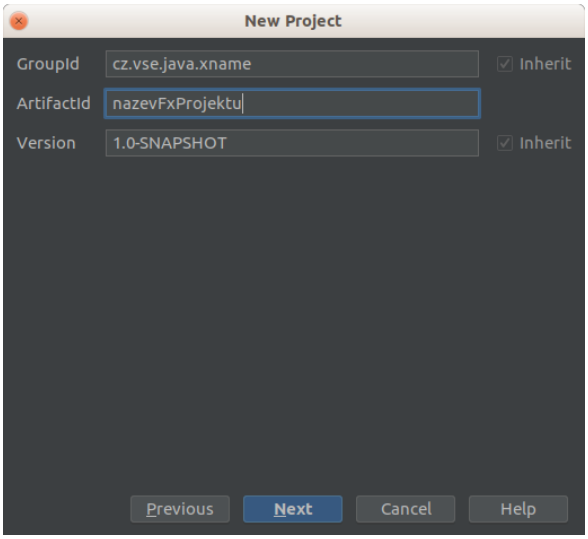
Obrázek 7.13: Úvodní obrazovka IntelliJ

2. V levém menu vybrat *Maven*.
3. Z rozbalovacího seznamu vybrat SDK 11.
4. Vybrat volbu *Create from archetype*.
5. Vybrat archetyp `org.openjfx:javafx-archetype-simple`. Pokud archetyp neexistuje, přidejte ho jako nový archetyp.



Obrázek 7.14: Vybrat javafx-archetype-simple

6. Zvolit SDK Javy, který chcete používat.
7. Potvrdit SDK a archetyp tlačítkem *Next*.
8. Vyplnit identifikační údaje nového Maven projektu (viz kapitola **Správa projektu > Identifikace Maven projektu**).
9. Potvrdit identifikační údaje tlačítkem *Next*.

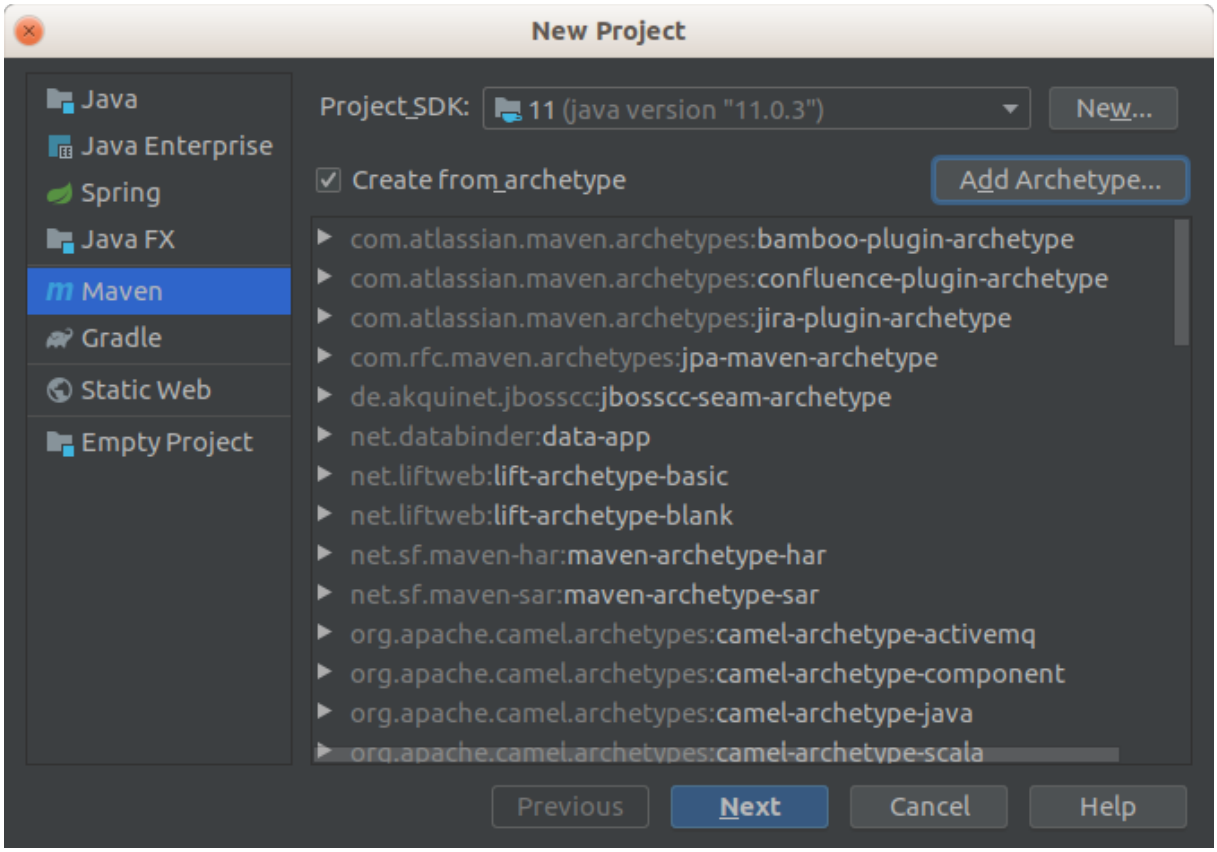


Obrázek 7.15: Identifikace nového projektu

7.3.2 Přidání archetypu, pokud neexistuje

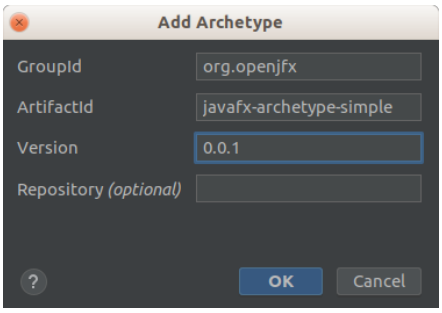
Pokud se v bodě 4 v seznamu archetypů nenacházel `org.openjfx:javafx-archetype-simple`, je třeba ho přidat jako **nový archetyp** následujícím způsobem:

4a. Vpravo nahoře kliknout na tlačítko *Add Archetype...*



Obrázek 7.16: Přidat archetyp

4b. Vyplnit údaje archetypu `org.openjfx:javafx-archetype-simple` dle Maven repozitáře.

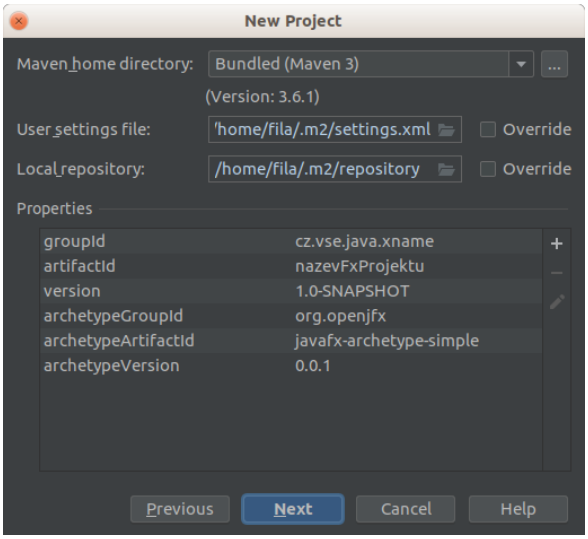


Obrázek 7.17: Přidat nový archetyp

7.3.3 Dokončení postupu

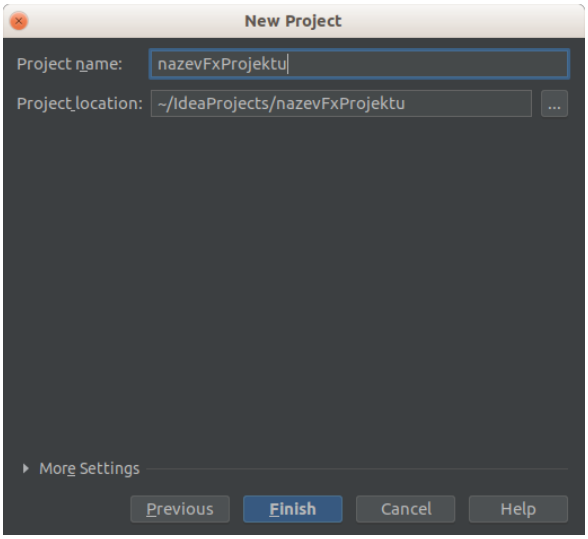
Po potvrzení identifikačních údajů nového Maven projektu (groupId, artifactId, version):

10. Potvrďte výchozí nastavení pro použití Maven (soubor s nastavením a umístění staženého kódu) a vlastností projektu.



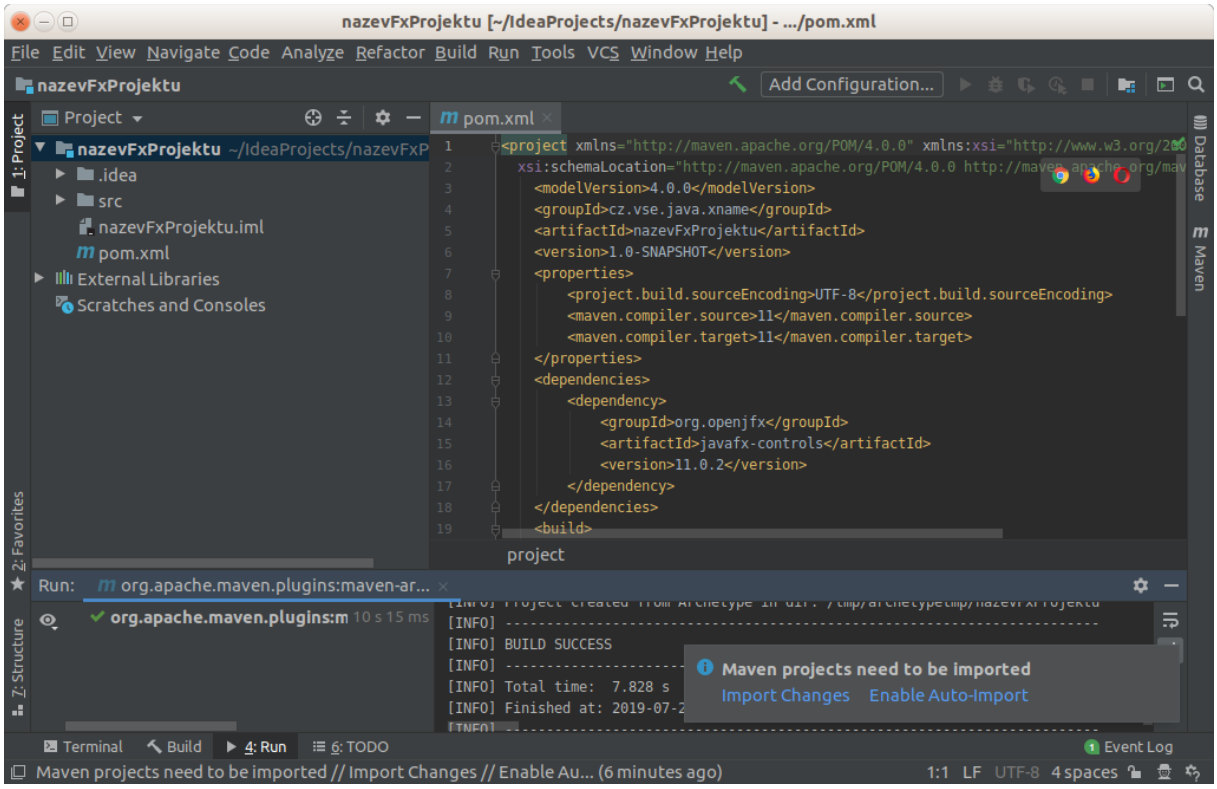
Obrázek 7.18: Vlastnosti nového projektu

11. V poslední obrazovce dialogu potvrďte název projektu a jeho umístění na disku.



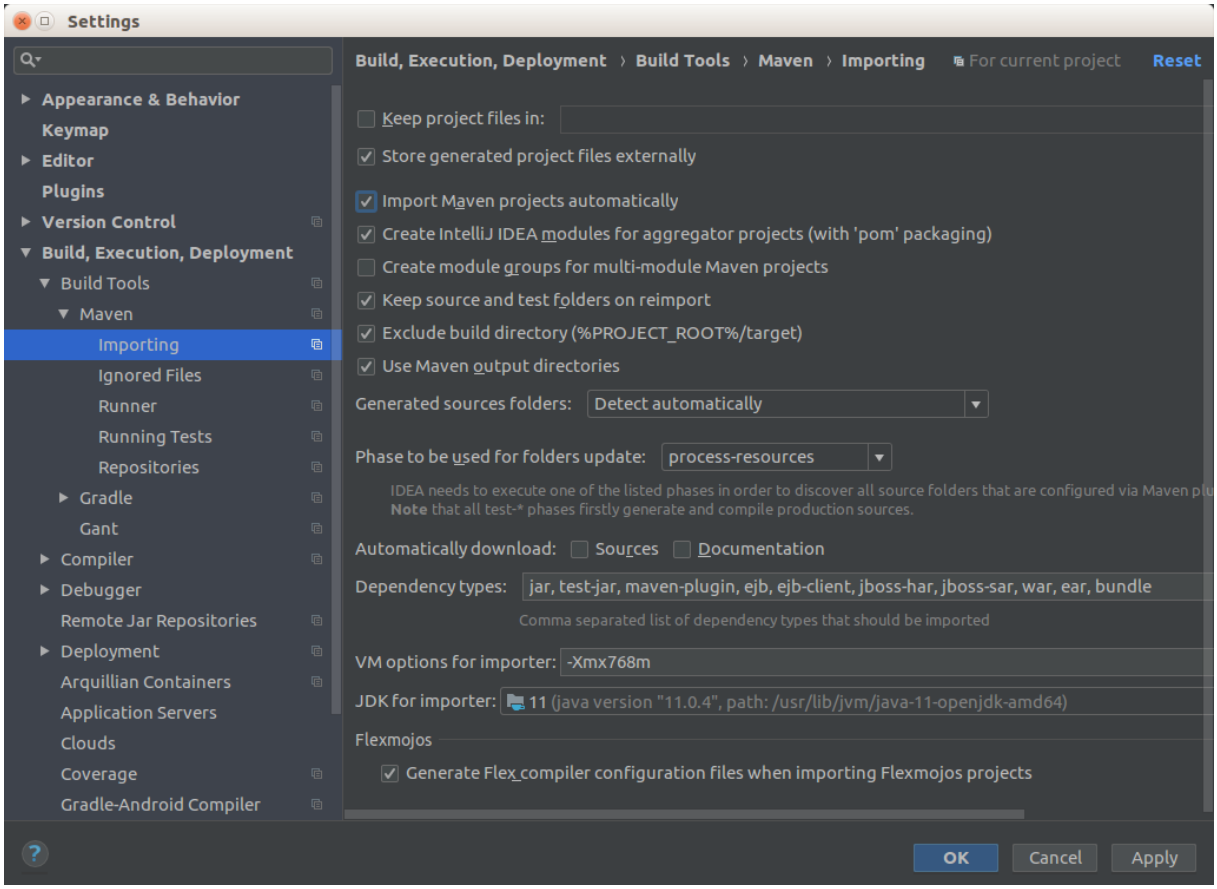
Obrázek 7.19: Název a umístění projektu

Projekt se vytvořil a zobrazila se struktura souborů a nově vytvořený objektový model (POM).



Obrázek 7.20: Vytvořený projekt

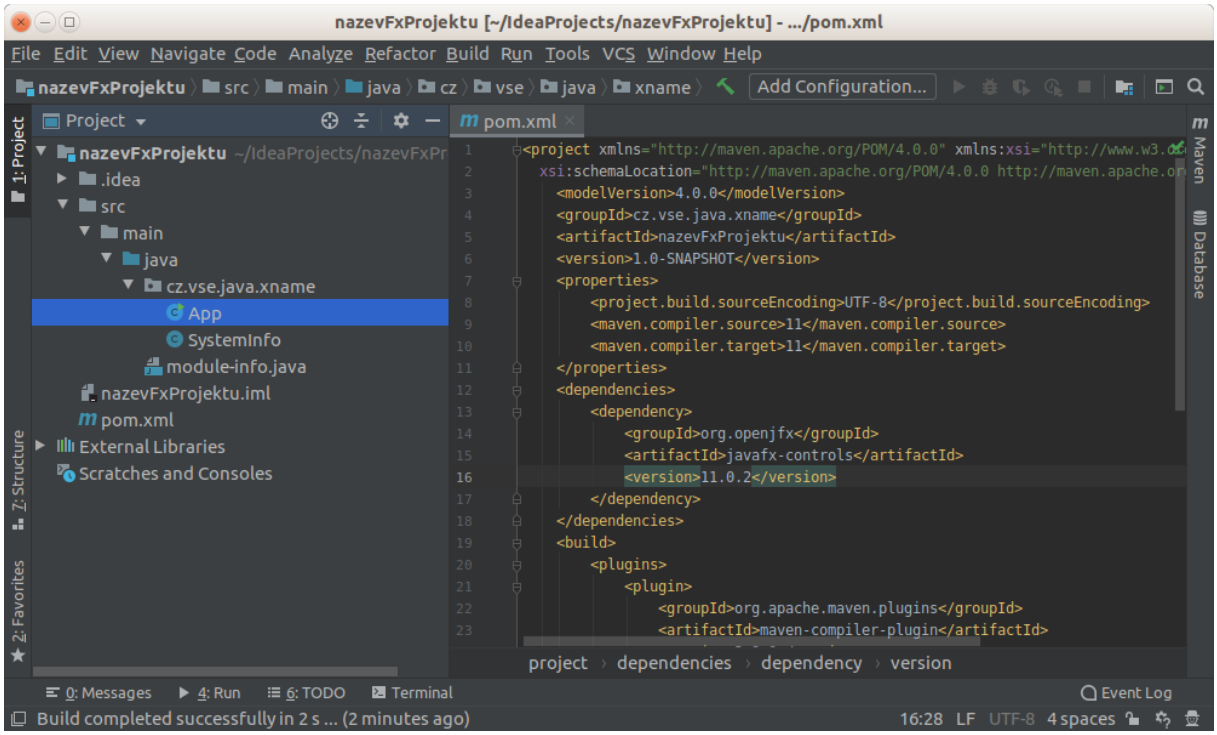
12. Pozornost věnujte malému informačnímu dialogu vpravo dole, kde je možné spravovat vlastnosti **importu**. Doporučené je kliknout na volbu *Enable Auto-Import*, která zaručí, že knihovny se budou stahovat automaticky při změně objektového modelu. Vlastnosti importu můžete upravovat i později v nastavení v dialogu *Build, Execution, Deployment > Build Tools > Maven > Importing* zaškrtnutím volby *Import Maven projects automatically*.



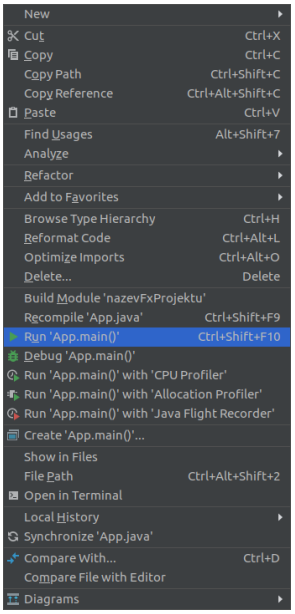
Obrázek 7.21: Nastavení importu

7.3.4 Spuštění projektu

Projekt se spustí kliknutím na spustitelnou třídu (s metodou *main*) pravým tlačítkem a vybráním volby *Run*.



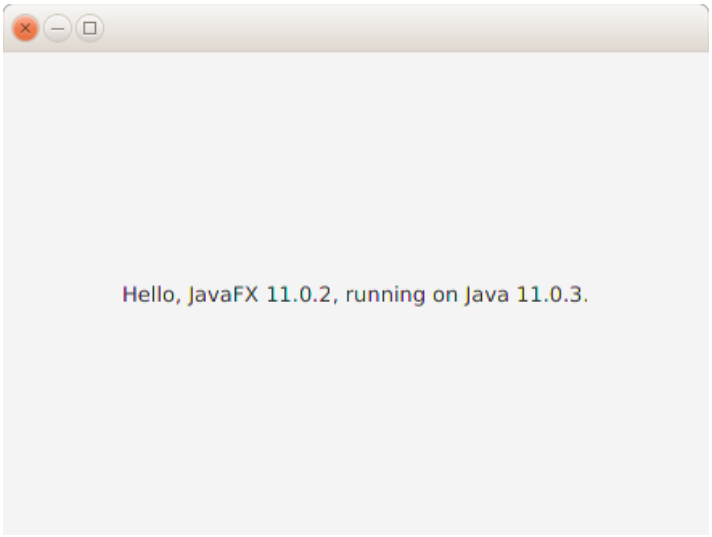
Obrázek 7.22: Spustitelná třída



Obrázek 7.23: Kontextové menu třídy

Alternativně je možné vybrat volbu *Run* z kategorie *Run* na horní liště.

Po spuštění se objeví okno s popisem verze JavaFX a JDK.



Obrázek 7.24: Spuštěný projekt

7.3.5 Možné problémy

Pokud IDEA nerozpoznává názvy tříd knihovny JavaFX, jednou z příčin může být:

- Neproběhlo správně stažení závislostí z Maven repozitáře a je potřeba zavolat ručně *Reimport All Maven Projects*. Volba se nachází v Maven panelu a má symbol dvou zelených šipek v kruhu (refresh symbol).
- Nevytvořila se konfigurace modulu. Vytvoření projektu z archetypu by mělo vytvořit modul a nakonfigurovat ho na použití s JavaFX knihovnou. Pokud se tak nestalo, je třeba řídit se návodem na konfiguraci u varianty D.

7.4 Maven projekt s OpenJDK 8 – varianta C

Podmínky pro uplatnění postupu:

- Je nainstalovaná Java OpenJDK verze 8.

Připojení knihovny JavaFX:

- Jelikož JavaFX není součástí OpenJDK 8 a v Maven repozitáři nejsou JavaFX knihovny pro verzi 8, je nutné knihovny JavaFX do projektu připojit ručně v konfiguraci projektu.

V případě, že vám podmínky neumožňují vytvořit projekt tímto způsobem, prozkoumejte další varianty.

7.4.1 Postup založení

Postup je stejný jako u *varianty A* s tím rozdílem, že je nutné mít nainstalovanou knihovnu JavaFX a případně nastavit ručně propojení s projektem.

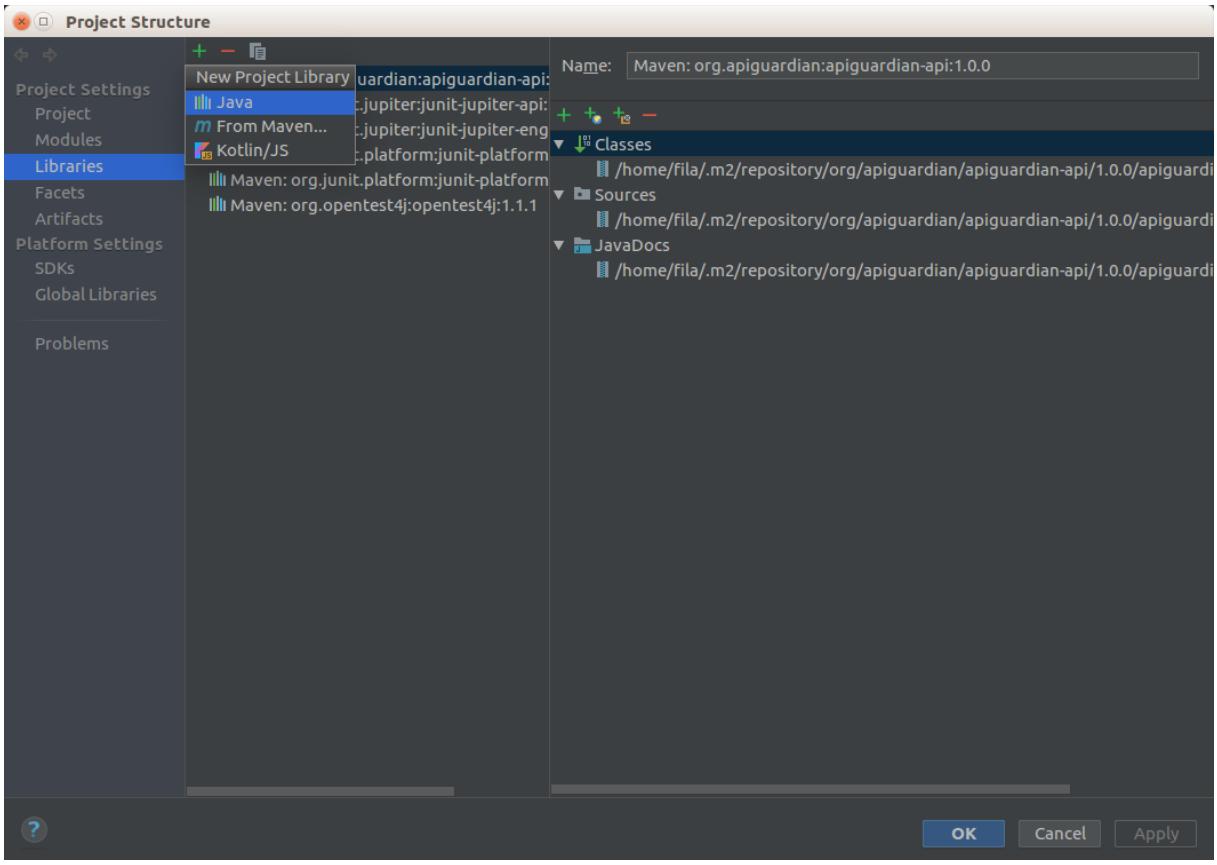
Podrobnosti o instalaci najdete v kapitole **JavaFX**.

7.4.2 Propojení JavaFX s projektem

Pokud **IDE neakceptuje třídy knihovny JavaFX**, je to známka buď nepovedené instalace JavaFX, nebo nutnosti propojit projekt s nainstalovanou knihovnou JavaFX na disku.

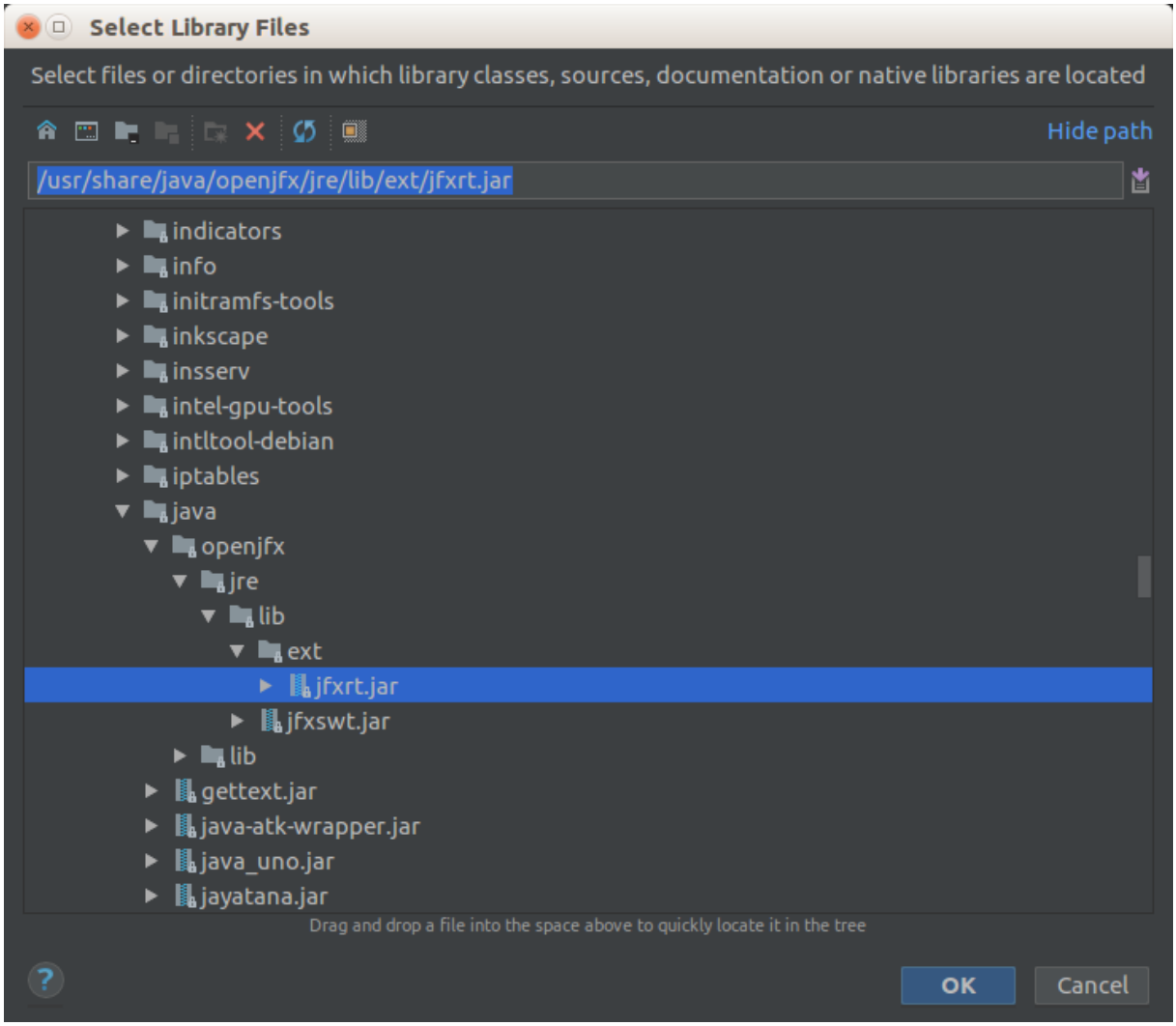
To je možné napravit tímto způsobem:

1. Otevřete dialog *Project Structure*, který je možné otevřít z nabídky *File > Project Structure*.
2. V dialogu na levé straně otevřete záložku ***Libraries***, která otevře přehled externích knihoven projektu.
3. Klikněte na **zelený symbol plus** nad levým seznamem knihoven a vyberte *Java*.



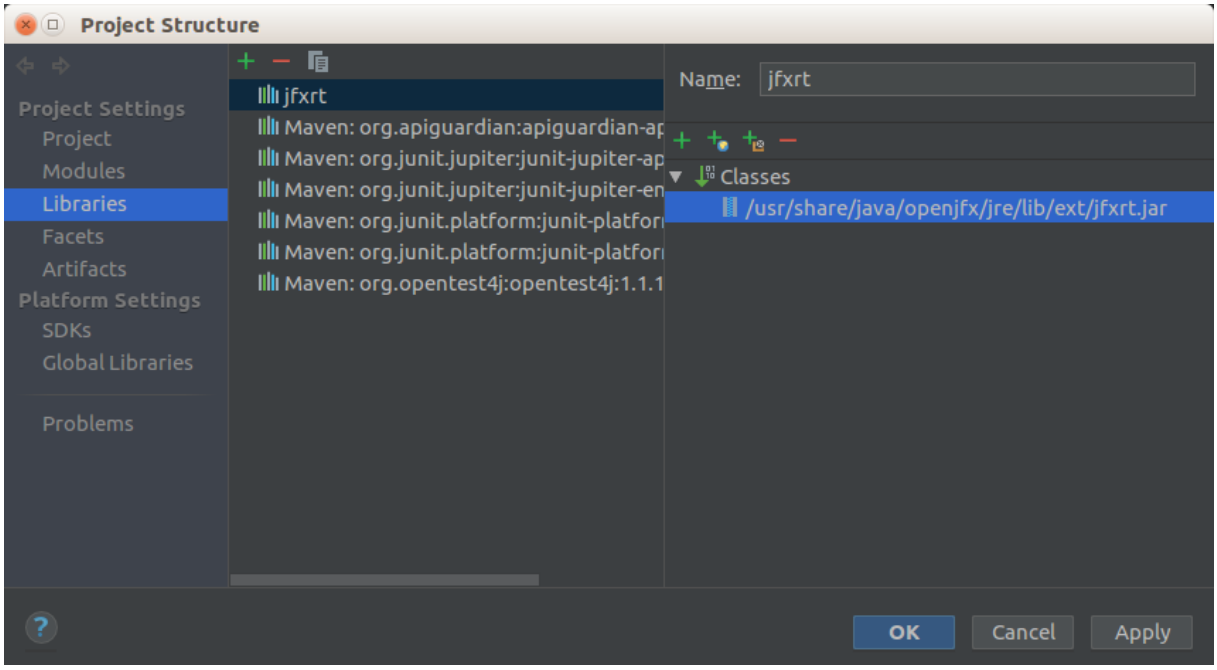
Obrázek 7.25: Přehled externích knihoven

4. V dialogu *Select Library Files* můžete na disku vyhledat umístění nainstalované knihovny JavaFX. Stačí přidat soubor ***jfxrt.jar***. Pokud nevíte přesně, kde soubor hledat, můžete použít příkaz `locate jfxrt.jar` (Linux).



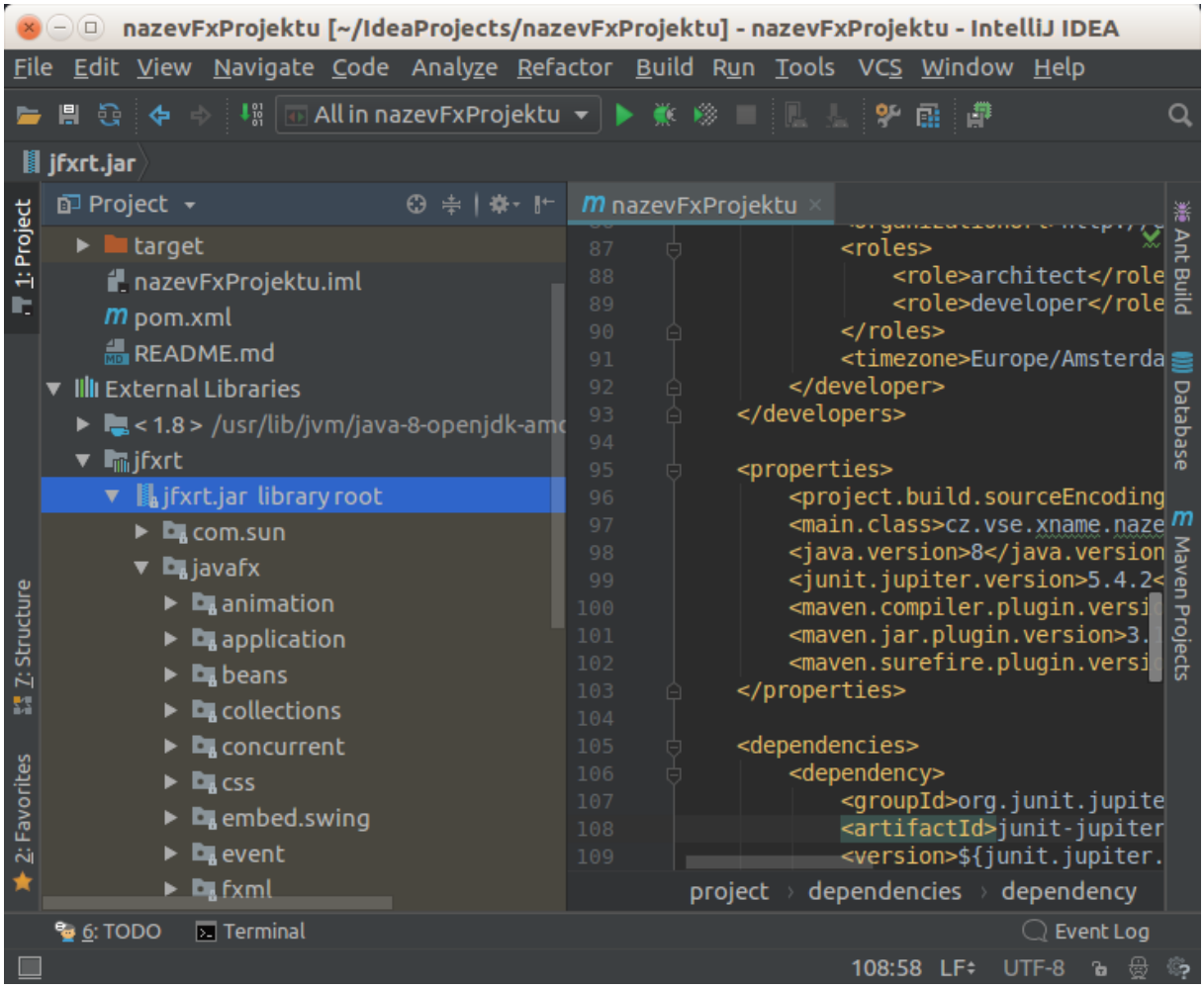
Obrázek 7.26: Výběr souborů knihovny

5. JavaFX by měla být vidět mezi ostatními knihovnami.



Obrázek 7.27: Přehled externích knihoven

6. V projektovém přehledu byste měli knihovnu vidět v sekci *External Libraries*, a to včetně jejího obsahu.



Obrázek 7.28: Přehled externích knihoven

7.5 Maven projekt s JDK 11 a ručním připojením JavaFX – varianta D

Podmínky pro uplatnění postupu:

- Je nainstalovaná Java OpenJDK verze 11.

Postup je uveden jen pro úplnost. Pokud chcete vyvíjet v SDK 11, je lepší připojit knihovny JavaFX jako Maven závislost (*varianta B*).

V případě, že vám podmínky neumožňují vytvořit projekt tímto způsobem, prozkoumejte další varianty.

7.5.1 Postup založení

Postup probíhá stejně jako v případě *variant A* a *C* s tím rozdílem, že při vytváření použijete **archetyp pro Javu 11**.

Doporučený archetyp `tech.raaf:java11-archetype:1.0.0`. Podrobnosti viz Maven repozitář.

Pokud je to nutné, po založení propojte s nainstalovanou knihovnou na disku jako u *varianty C*. Knihovna musí být nainstalovaná (viz **kapitola JavaFX**).

Nakonec je třeba **nakonfigurovat modul**. Modul lze konfigurovat jen tehdy, pokud je projekt v IDEA nastaven nejen na SDK 11, ale také jazyková pravidla Javy verze 11. Modul se nachází ve projektovém stromu v `src > main > java` a nese název `module-info.java`. Modul by měl obsahovat minimálně tento kód, přičemž `nazevModulu` představuje plný název hlavního balíčku aplikace (např. `cz.vse.java.xname.nazevfxaplikace`).

```
module nazevModulu {
    requires javafx.fxml;
    requires javafx.controls;
    exports nazevModulu;
}
```

7.6 IDEA projekt pomocí průvodce – varianta E

Postup platí pro použití s Java SDK 8 i 11. Využívá průvodce IntelliJ IDEA pro nový JavaFX projekt.

Připojení JavaFX:

- V případě Java OpenJDK 8 nebo obou verzí 11 může být nutné provést ruční propojení s knihovnou JavaFX.
- Po případném převedení na Maven projekt je možné u SDK 11 připojit knihovnu JavaFX jako závislost.

Pokud projekt není později převeden na Maven, je veškeré nastavení projektu uložené jen v souborech IDE, což může způsobit problémy při sdílení projektu dalším osobám.

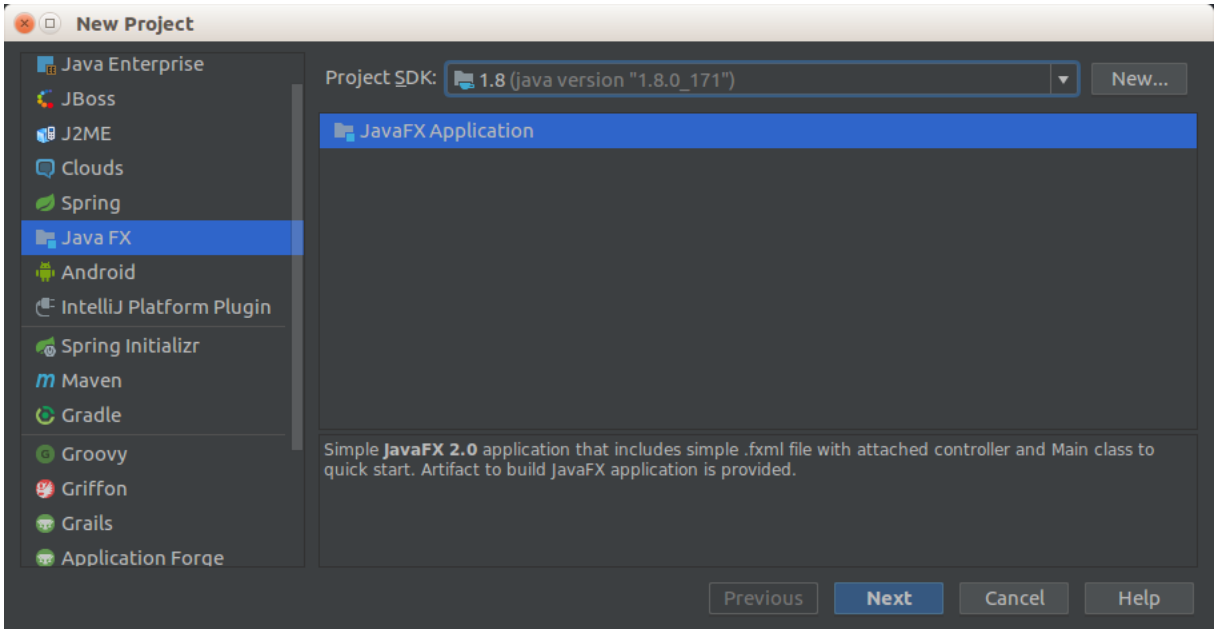
V případě, že vám podmínky neumožňují vytvořit projekt tímto způsobem, prozkoumejte další varianty.

7.6.1 Postup založení

1. Na úvodní obrazovce zvolit *Create New Project*.

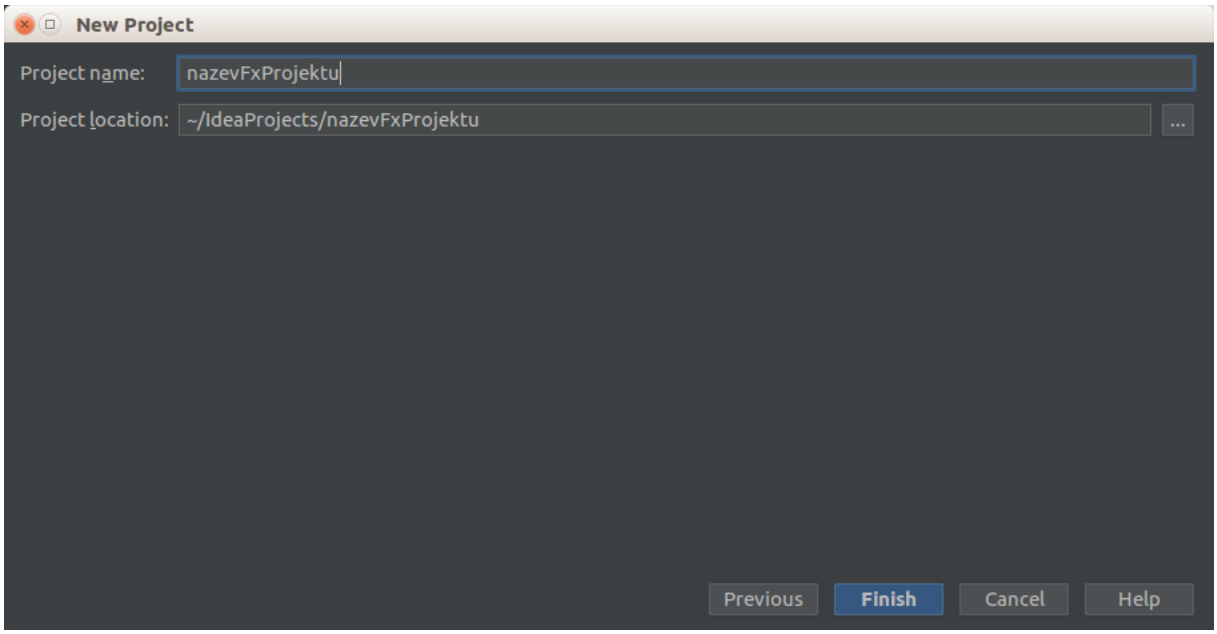
Obrázek 7.29: Úvodní obrazovka IntelliJ

2. V seznamu nalevo vybrat *JavaFX*.
3. Vybrat požadovanou verzi **Java SDK**, ve které chcete projekt vytvářet.
4. Kliknout na tlačítko *Next*.



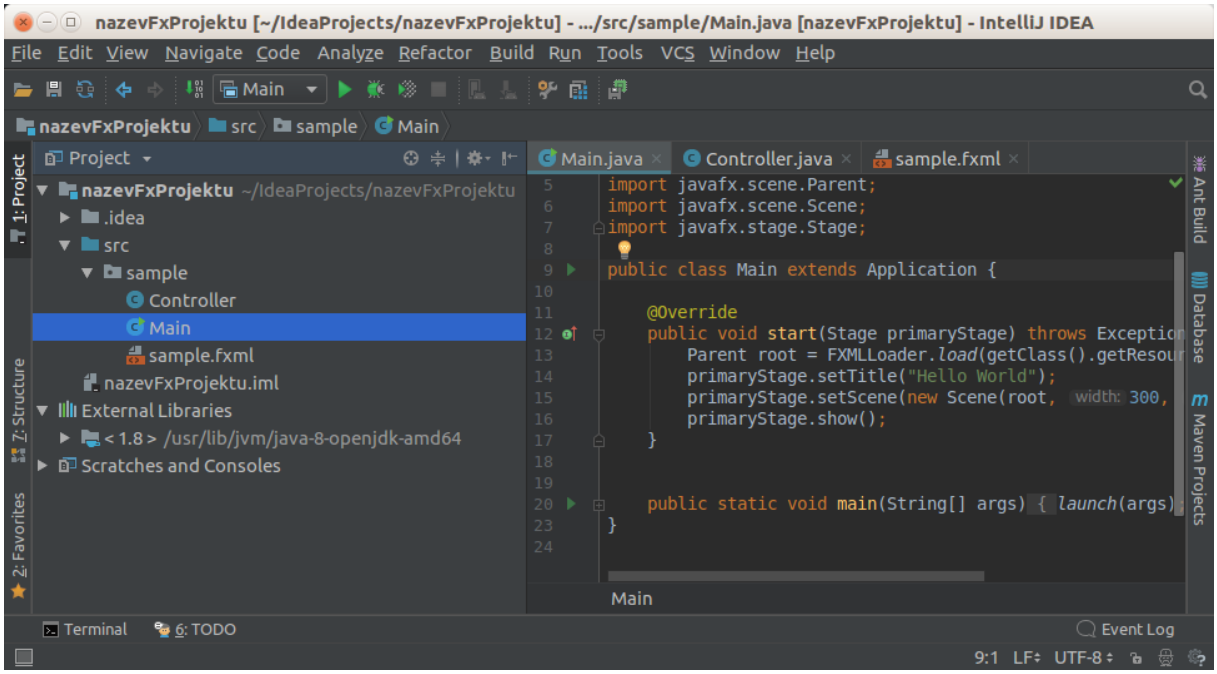
Obrázek 7.30: Průvodce novou JavaFX aplikací

5. Pojmenovat projekt a případně upřesnit jeho umístění.



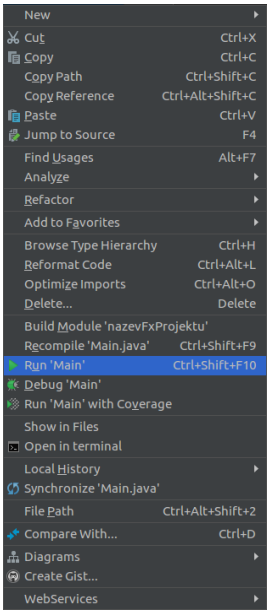
Obrázek 7.31: Zadání jména a umístění projektu

6. Projekt se vytvořil se vzorovým balíčkem *sample*, spouštěcí třídou *Main*, prázdným kontrolerem a prázdným FXML.



Obrázek 7.32: Vytvořená JavaFX aplikace

7. Aplikaci spustit pomocí kliknutí pravým tlačítkem na třídu *Main*, která obsahuje metodu *main*.
8. Z kontextového menu vybrat volbu *Run 'Main'*.



Obrázek 7.33: Spuštění aplikace přes kontextové menu

9. Výsledkem by mělo být prázdné okno s názvem *Hello World*.

Obrázek 7.34: Spuštěná aplikace

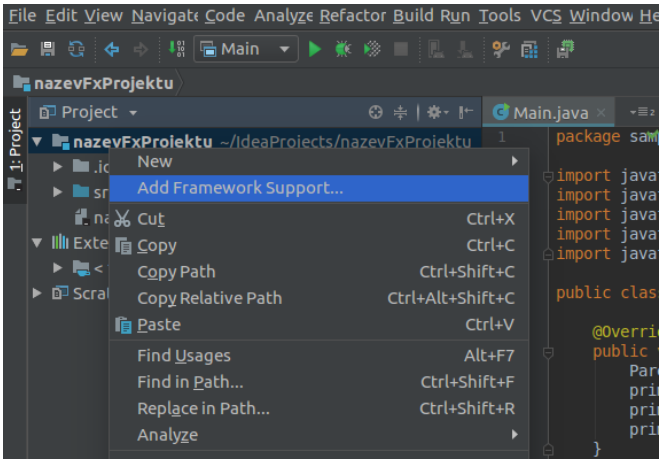
V případě, že projekt nelze spustit, není vhodně nainstalovaná JavaFX nebo vyžaduje propojení na projekt jako externí knihovna (viz varianta C a D). Pokud používáte Javu 11, je možné použít knihovnu JavaFX jako závislost po převedení na Maven projekt.

7.6.2 Převedení na Maven projekt

Pokud chcete, aby se projekt dal dobře sdílet jiným vývojářům, je doporučené převést ho na Maven nebo Gradle projekt.

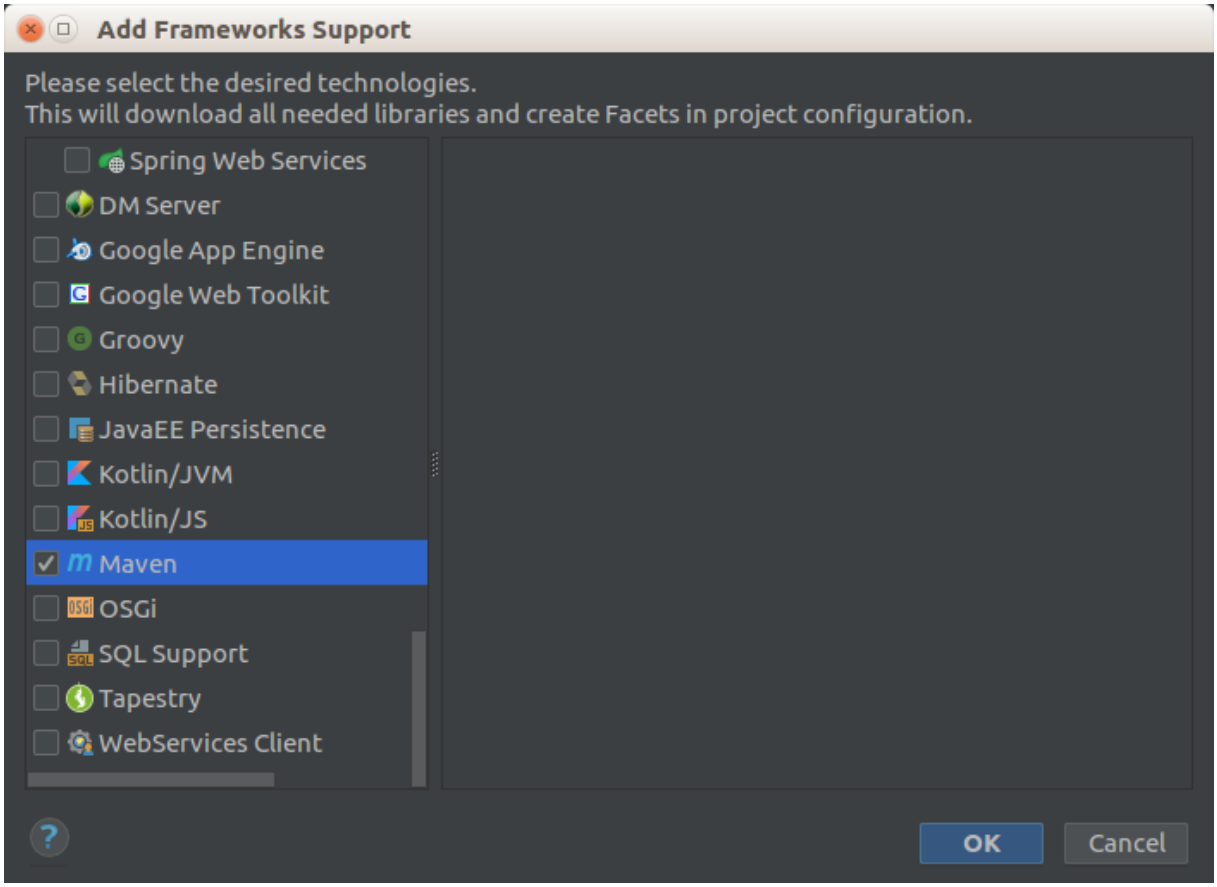
V IntelliJ IDEA je možné každý projekt převést i později na Maven projekt následujícím postupem:

1. Pravým tlačítkem klikněte na název projektu v panelu se strukturou projektu.
2. V kontextovém menu vyberte *Add Framework Support...*



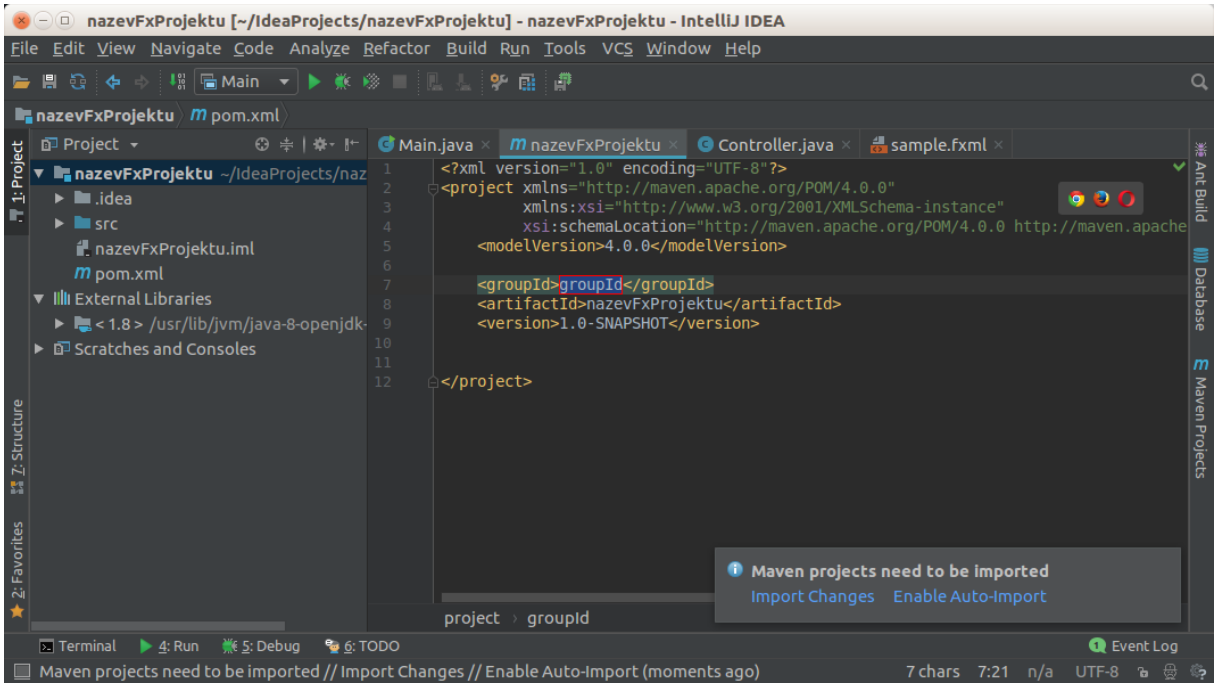
Obrázek 7.35: Přidat podporu frameworku pro projekt

3. V seznamu frameworků zaškrtněte *Maven*.



Obrázek 7.36: Přidat podporu pro Maven

4. Projekt se převedl na Maven projekt a vytvořil se objektový model POM.
5. Pozornost věnujte nastavení **importu** a zapněte volbu *Enable Auto-Import*.



Obrázek 7.37: Převedený projekt

5. V objektovém modelu upravte identifikační údaje. *ArtifactId* se přenesl z nastavení IDEA. Stačí tak vyplnit *groupId* na `cz.vse.java.xname`.
6. Doplňte objektový model o obvyklé nastavení pluginů a závislostí (viz ostatní varianty).
7. Po převedení projekt nemusí jít spustit z důvodu jiné struktury složek, než odpovídá konvencím Maven projektu. Pro nápravu je potřeba buď nakonfigurovat objektový model na stávající strukturu, nebo strukturu změnit, aby odpovídala konvencím (viz **kapitola o Maven**).

Kapitola 8

Sdílení projektu na vzdálený repozitář

V předchozí kapitole jsme vytvořili nový JavaFX projekt, jehož obsah je náchylný k nevratným změnám a je dostupný jen lokálně. Každý takový projekt je třeba zahrnout do systému pro správu verzí a propojit se vzdáleným repozitářem. Do vzdáleného repozitáře se projekt jednak zálohuje a jednak je dostupný dalším spolupracovníkům.

Pro dokončení postupu v této kapitole je nezbytné pochopit princip práce se systémem na správu verzí Git, kterému se věnuje kapitola Správa verzí.

Pro sdílení projektu na vzdálený repozitář je třeba se držet následujícího postupu:

1. inicializace lokálního repozitáře v adresáři projektu,
2. výběr souborů pro zahrnutí do revize a odeslání změn,
3. založení prázdného vzdáleného repozitáře,
4. propojení lokálního a vzdáleného repozitáře,
5. odeslání lokálních změn na vzdálený repozitář.

Pro splnění kroků můžete využít různé nástroje. Kapitola se blíže věnuje použití příkazové řádky, IntelliJ IDEA a GitKraken.

Jako vzdálený repozitář budeme používat *GitLab.com*. GitLab je open source platforma určená pro podporu vývoje aplikací. GitLab, spolu s dalšími alternativami, blíže představila kapitola Vzdálené repozitáře.

8.1 Inicializace lokálního repozitáře v adresáři projektu

8.1.1 Inicializace v příkazové řádce

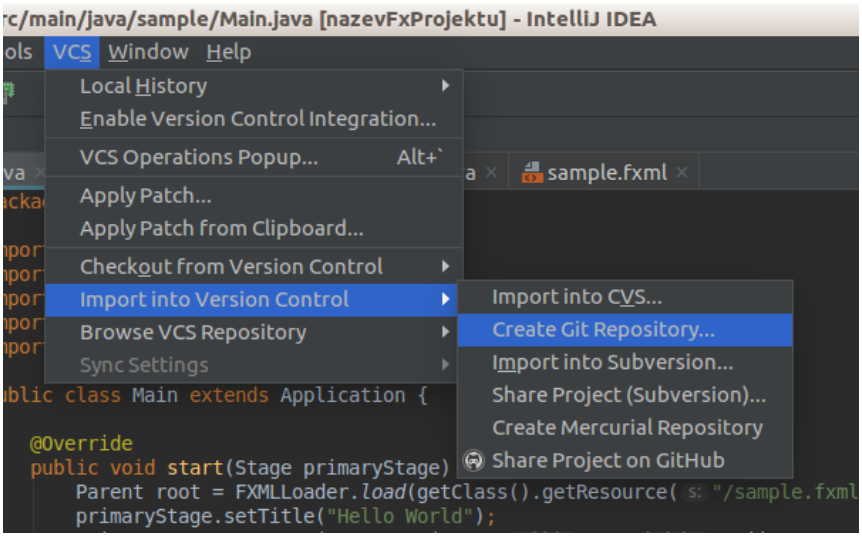
V příkazové řádce lze inicializaci provést příkazem `git init`. Příkaz je potřeba zavolat v kořenovém adresáři projektu. Inicializaci ověříte zadáním `git status`.

Výpis by měl vypadat takto:

```
On branch master

Initial commit

Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .idea/
    nazevFxProjektu.iml
    pom.xml
    src/
    target/

nothing added to commit but untracked files present (use "git add" to track)
```

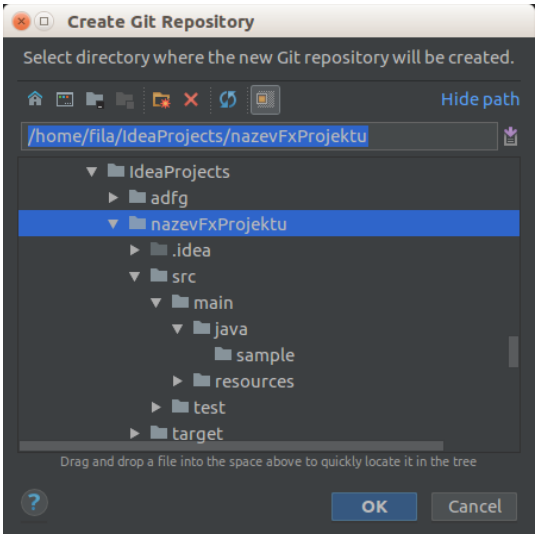
8.1.2 Inicializace v IntelliJ IDEA

Výběrem z nabídky *VCS > Import into Version Control > Create Git Repository...* v prostředí IntelliJ IDEA inicializujete lokální repozitář.



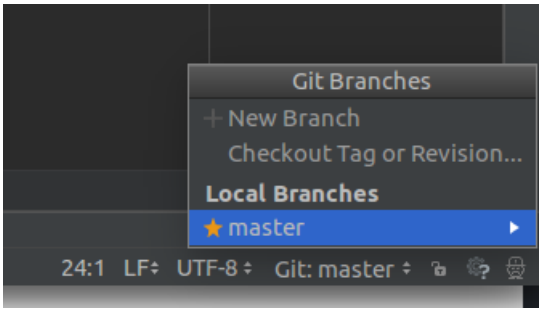
Obrázek 8.1: Inicializace lokálního repozitáře

V dialogu vyberte kořenový adresář projektu. Obvykle nese stejný název jako projekt a měl by se v něm na první úrovni nacházet adresář *src*.



Obrázek 8.2: Výběr umístění pro inicializaci repozitáře

Správnou inicializaci lokálního repozitáře poznáte podle toho, že se v IDE obvykle změní barva názvů souborů v projektovém stromu. V IntelliJ IDEA se v pravém spodním rohu objeví informace o tom, na které jste aktuálně větvi.



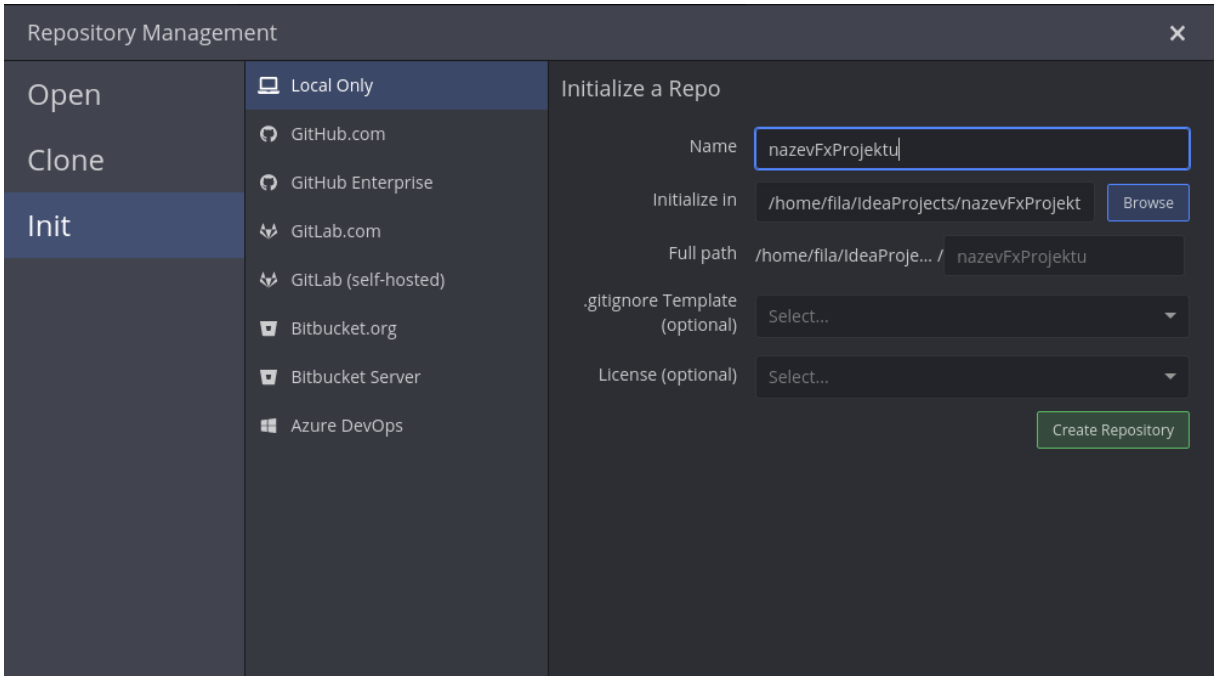
Obrázek 8.3: Zobrazení aktuální větve

8.1.3 Inicializace v GitKraken

GitKraken je grafický klient pro správu verzí pomocí systému Git. Odkaz na stažení spolu s dalšími alternativami uvádí kapitola Instalace Gitu.

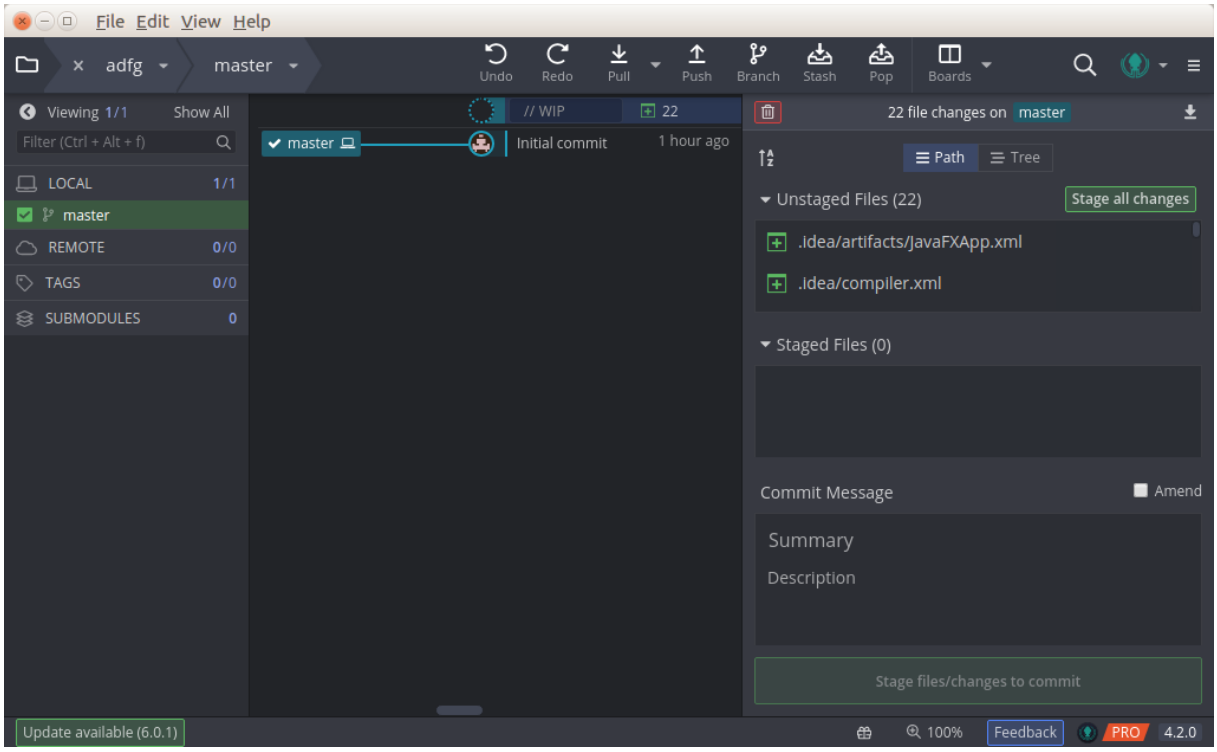
V GitKraken se nový repozitář inicializuje volbou *File > Init Repo*.

V dialogu, který se otevře, je potřeba nazvat projekt a zvolit, kde se nachází. Pole *Initialize in* by mělo obsahovat o úroveň vyšší adresář, než je ten kořenový. Kořenový adresář se doplní podle názvu projektu. Před samotným vytvořením si raději zkontrolujte *Full path*.



Obrázek 8.4: Inicializace repozitáře v prostředí GitKraken

Po inicializaci uvidíte základní obrazovku prostředí GitKraken se zobrazením inicializačního commitu. Inicializační commit je prvním záznamem o verzi. Bez něho nejde lokální repozitář v GitKrakenu otevřít.



Obrázek 8.5: Dokončená inicializace repozitáře v prostředí GitKraken

8.2 Výběr souborů pro zahrnutí do revize a odeslání změn

8.2.1 Výběr souborů pro správu verzí

V tomto bodě se vyplatí zvážit, které soubory je vhodné sdílet do repozitáře. Obecně platí zásada, že se sdílí **jen zdrojový kód**, případně další zdroje jako např. obrázky nebo HTML.

Vhodné je sdílet nastavení projektu, která jsou **nezávislá na IDE**, např. Maven (*pom.xml*) nebo Gradle, ze kterých se projekt po naklonování snáze nastaví.

Při sdílení projektu **bez Maven nebo Gradle** je třeba zvážit možnosti importu dle každého IDE. Například pro IntelliJ IDEA je třeba držet se rad v článku oficiální podpory.

Soubory, které nemají být předmětem verzování, je možné nastavit v nástroji na správu verzí (IDE, GitKraken...), případně přímo v souboru `.gitignore` (viz git-scm.com/docs/gitignore), do kterého se nastavení zapisuje.

Soubor `.gitignore` by se měl nacházet v **kořeni projektu**. Příklad souboru, ve kterém nesledujeme a nesdílíme změny nastavení IDE (adresář `.idea` a soubor s příponou `.iml`) a výstupy kompilace a sestavení aplikace (složka `target`), by mohl vypadat takto:

```
.idea/

*.iml

target/
```

V souvislosti se souborem `.gitignore` bude potřeba vyřešit otázku, zda by měl být soubor také předmětem verzování a sdílen tak dalším vývojářům skrze vzdálený repozitář. Na otázku není jednotný pohled, nicméně vývojáři na StackOverflow **doporučují `.gitignore` sdílet**. V případě, že vznikne individuální potřeba pro ignorování dalších souborů, je možné toto upravit v globálním nastavení Gitu.

8.2.2 Výběr souborů v příkazové řádce

1. V oblíbeném editoru otevřete soubor `.gitignore` umístěný v kořeni projektu nebo jej vytvořte (např. příkazem `nano .gitignore`).
2. Do řádků napište relativní cesty k souborům, které mají být vynechány. Pro zápis lze použít zástupný znak `*` nebo `**` (viz dokumentace).
3. Po vytvoření `.gitignore` by v projektu měl zůstat k verzování jen zdrojový kód a popřípadě objektový model `pom.xml` a `.gitignore`. Příkaz `git status` by měl vrátit obdobu toho výstupu:

```
On branch master
Untracked files:
  (use "git add <file>..." to include in what will be committed)

    .gitignore
    pom.xml
    src/

nothing added to commit but untracked files present (use "git add"
to track)
```

4. Zadáním příkazu `git add -A` se všechny změny převedou do seznamu změn k odeslání (*stage*) a připraví k zápisu. Přepínač `-A` zaznamená všechny nové, změněné nebo smazané soubory. Kromě přepínače `-A` je možné použít `git add -u` pro zaznamenání změn pouze na existujících souborech. Podrobnosti najdete v dokumentaci příkazu *add*.

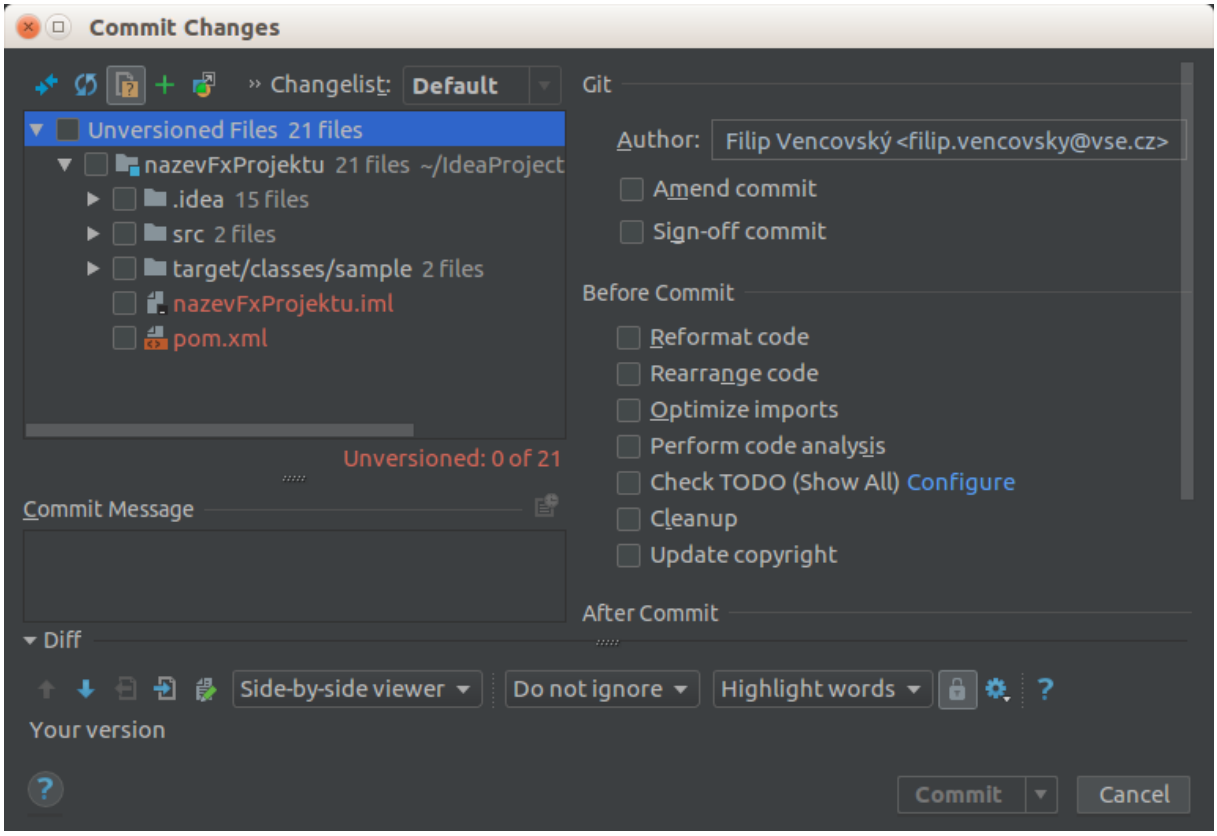
```
On branch master
Changes to be committed:
  (use "git reset HEAD <file>..." to unstage)

    new file:   .gitignore
    new file:   pom.xml
    new file:   src/main/java/cz/vse/xname/nazevFxProjektu/HomeController.java
    new file:   src/main/java/cz/vse/xname/nazevFxProjektu/Main.java
    new file:   src/main/resources/HomeController.fxml
```

5. Změny odešlete jako revizi příkazem `git commit -m "nahrání souborů"`. Přepínač `-m` není povinný, ale umožní vám zadat název revize v jednom kroku. Bez přepínače `-m` se otevře výchozí textový editor, ve kterém je možné název revize také napsat. Názvy revizí je vhodné volit krátké a výstižné vzhledem k odesílaným změnám.
6. Po úspěšném nahrání zobrazí `git status`, že adresář je beze změn.

```
On branch master
nothing to commit, working directory clean
```

8.2.3 Výběr souborů v IntelliJ IDEA

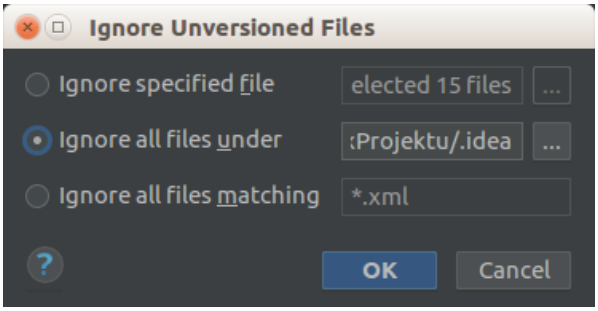
V IntelliJ IDEA se změny do revize zahrnou v dialogu s názvem *Commit Changes*, který můžete otevřít z hlavního menu volbou *VCS* > *Commit...*. Dialog na levé straně zobrazuje seznam nových nebo modifikovaných souborů. Pod seznamem je textová oblast pro zadání názvu změny (*Commit Message*).



Obrázek 8.6: Dialog k tvorbě revize v IntelliJ IDEA

Seznam souborů zahrnuje i ty, které nejsou doporučené pro verzování (nastavení workspace, zkompilované třídy). Proto je potřeba je vyjmout ze sledování verzí.

Vyjmout soubor nebo složku ze sledování můžete po kliknutí pravým tlačítkem na soubor. Zobrazí se dialog *Ignore Unversioned Files*, ve kterém je možné zvolit, zda mají být předmětem vynětí jednotlivé soubory (*specific file*), obsah složky (*files under*) nebo soubory vyhovující masce (*files matching*).



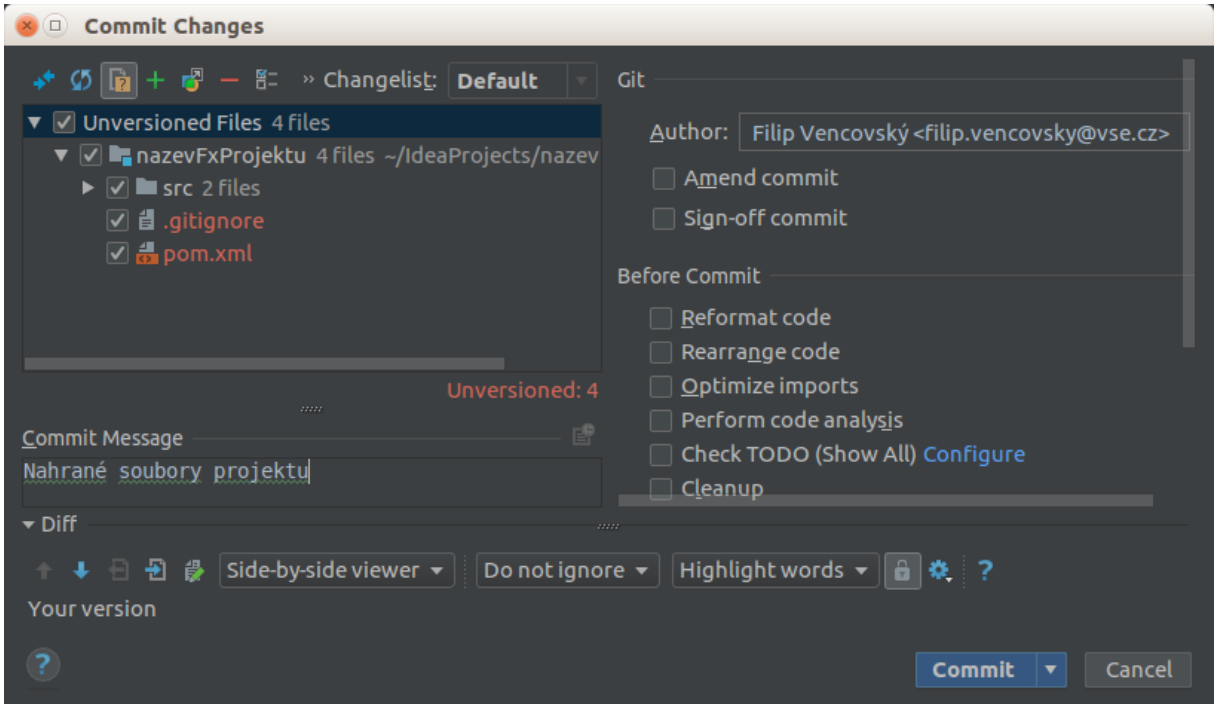
Obrázek 8.7: Vynětí souborů ze sledování změn v IntelliJ IDEA

Takto vyňaté soubory již dále nejsou předmětem sledování změn v IntelliJ IDEA, ale v jiných klientech pro správu verzí mohou být namísto stavu *ignored* ve stavu *unversioned*.

Je to způsobené tím, že IntelliJ IDEA **nevytváří** `.gitignore`. Namísto toho soubor, složku nebo masku souborů uloží do nastavení (*.idea/workspace.xml*). K nastavení lze přistoupit otevřením dialogu *File > Settings > Version Control > Ignored Files*.

Z důvodu špatné přenositelnosti seznamu ignorovaných souborů z nastavení IDEA **není toto řešení vhodné** a je lepší vytvořit `.gitignore`, viz kroky 1 a 2 v předchozí kapitole Výběr souborů v příkazové řádce.

Pro odeslání revize je nutné vybrat požadované soubory: zdrojový kód, `.gitignore` a potažmo i Maven konfigurace. Zkontrolujte, zda je vyplněné pole *Author*, a zvolte dostatečně vysvětlující název revize (*Commit Message*). Volby na pravé straně dialogu mohou zůstat nazaškrtnuté. Připravenou revizi odešlete tlačítkem *Commit*.

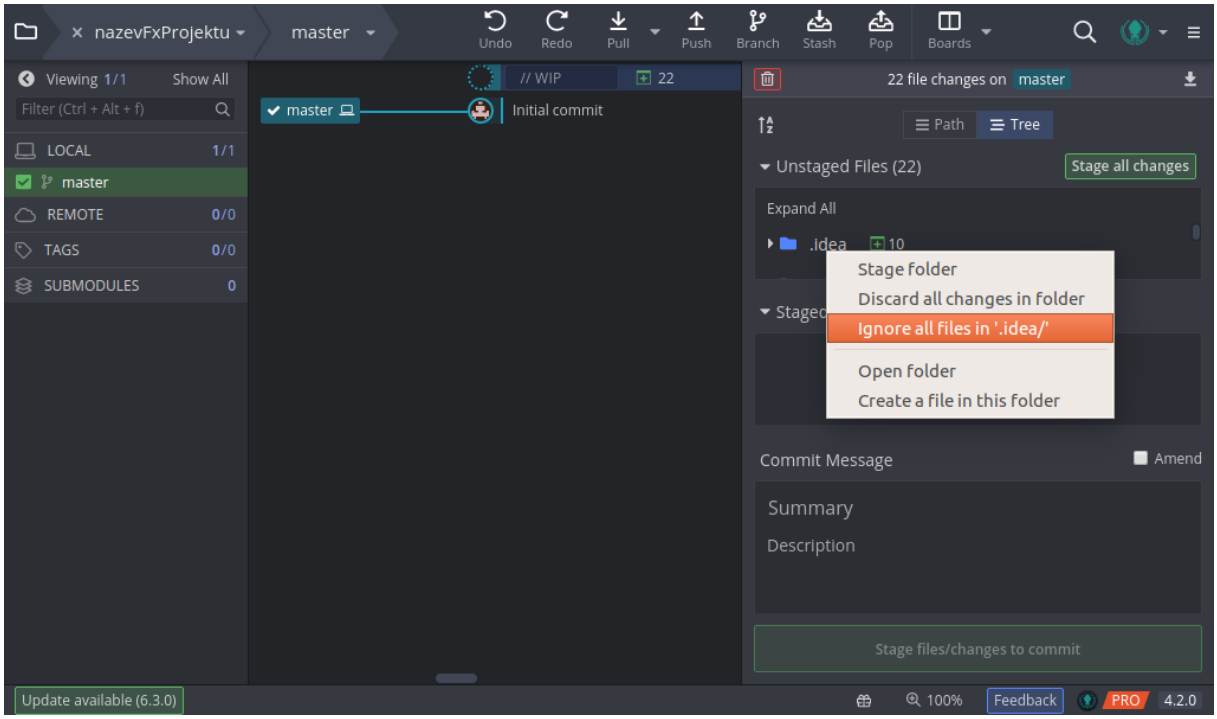


Obrázek 8.8: Odeslání změn ve formě revize v IntelliJ IDEA

8.2.4 Výběr souborů v GitKraken

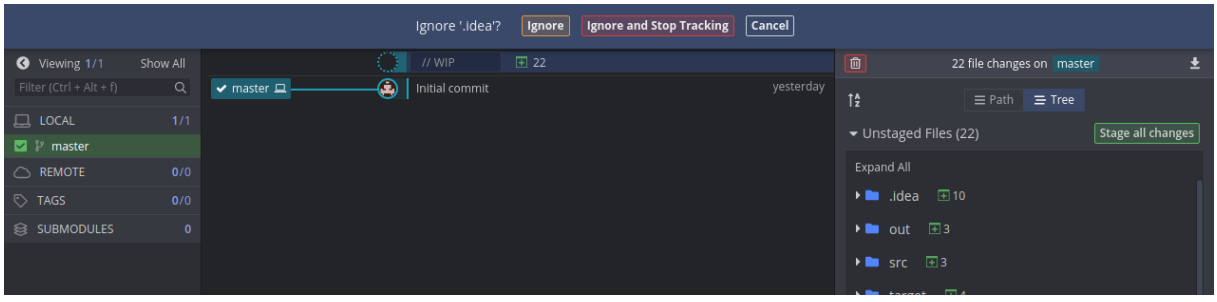
Přidání souborů do správy verzí a ignorování souborů pomocí nástroje GitKraken najdete skvěle zpracované v návodu na oficiálních stránkách.

Nejjednodušší cesta, jak vyjmout soubory ze sledování změn v GitKraken, je volba *Ignore all files in*, která se zobrazí po kliknutí pravím tlačítkem na složku nebo soubor v sekci *Unstaged files*. Vytvoří se tím soubor `.gitignore`, pokud neexistuje, a položka se přidá na nový řádek. Výsledek je stejný jako při ruční konfiguraci souboru `.gitignore` (kroky 1 a 2 v kapitole Výběr souborů v příkazové řádce).



Obrázek 8.9: Vynětí souborů ze sledování změn v GitKraken

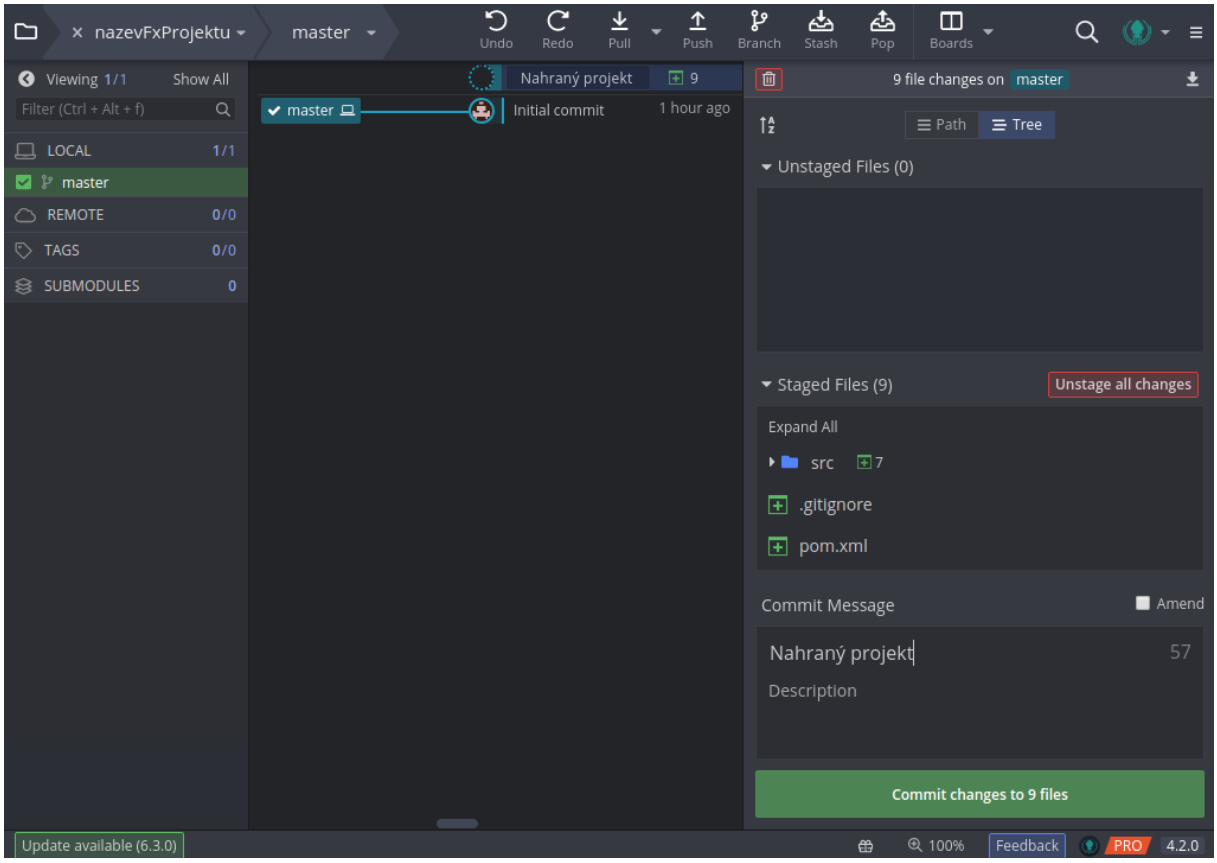
Pokud chcete ignorovat soubor, který byl již předmětem sledování změn v minulosti, klikněte pravým tlačítkem na soubor v nabídce *Unstaged Files* a zvolte *Ignore*. Po potvrzení volby kontextového menu se objeví v horní části okna dialog s volbami *Ignore* a *Ignore and Stop Tracking*. Volba *Ignore* vloží záznam do souboru `.gitignore`, ale změny budou stále viditelné. Druhá volba navíc vymaže soubor z indexu repozitáře.



Obrázek 8.10: Dialog k vynětí ze sledování změn v GitKraken

Zapsání souborů do lokálního repozitáře provedeme následujícím způsobem. Požadované soubory a složky přidáme do *stage* přesunutím z oblasti *Unstaged Files* do *Stage Files*. Přesun lze provést tlačítkem *Stage file*, které se objeví vedle každého souboru, nebo tlačítkem *Stage all changes*, které přesune všechny nové, změněné nebo odebrané soubory (viz přepínač `-A` v dokumentaci).

Po přesunutí souborů do *stage* je možné takto vybrané soubory zaznamenat jako revizi velkým zeleným tlačítkem *Commit changes*. Předtím je ale nutné nastavit jméno revize (*commit message*), které bude dobře vystihovat změnu v projektu.



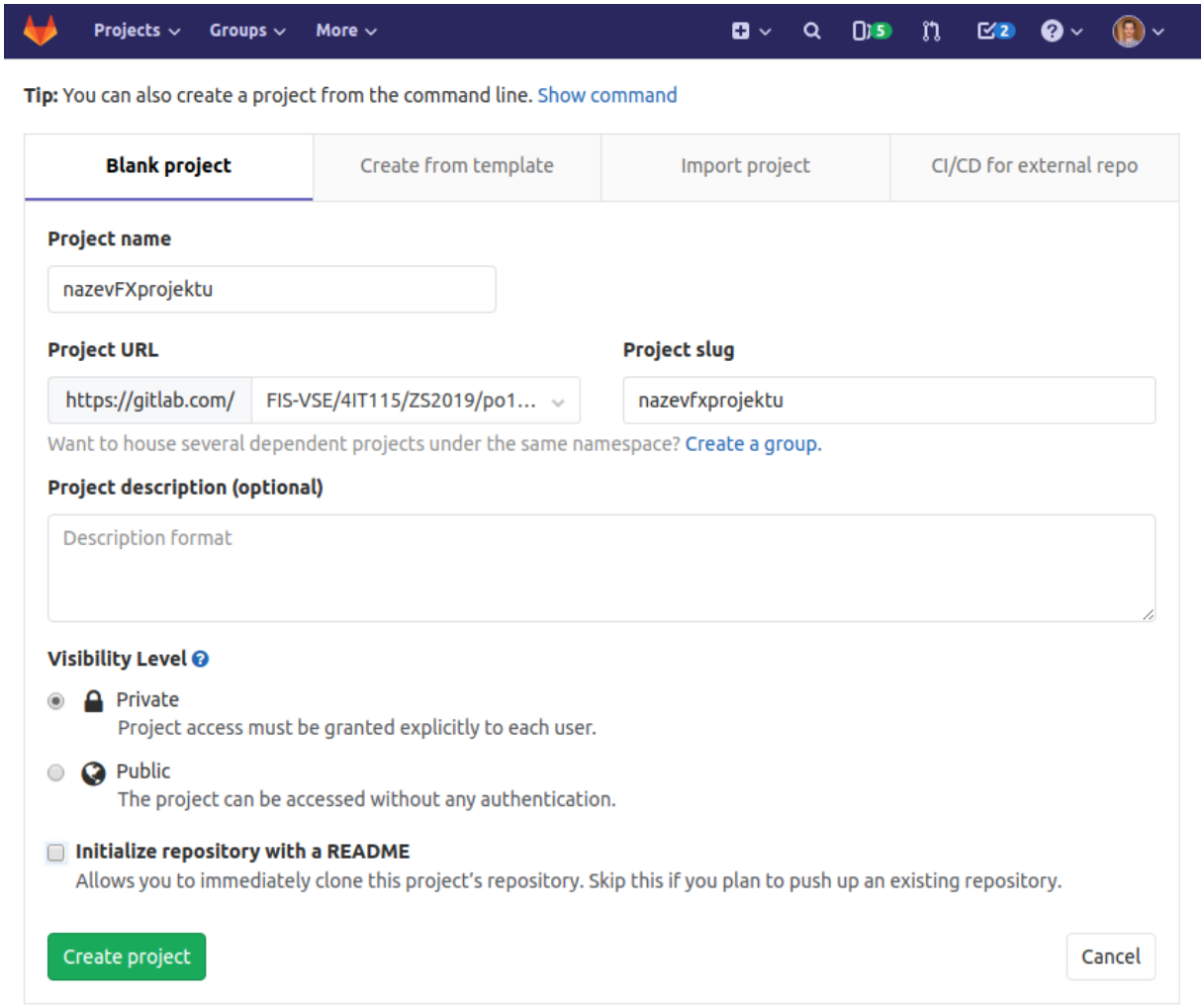
Obrázek 8.11: Změny připravené k odeslání do lokálního repozitáře v GitKraken

8.3 Založení prázdného vzdáleného repozitáře

Před nahráním do vzdáleného repozitáře je třeba založit nový projekt na GitLab.com (nebo jiném úložišti dle preference). Nový projekt založíte zeleným tlačítkem *New Project* na úvodní stránce nebo ve vybrané skupině projektů (např. pro vaše cvičení).

Při zakládání vzdáleného repozitáře vždy zvažte, jestli do něj plánujete nasdílet kód z existujícího repozitáře, nebo ho naklonovat jako prázdný a nové soubory do něj vložit (viz podkapitola Alternativní postup sdílení projektu).

Pokud plánujete, v souladu s dosavadním postupem v této kapitole, napojit na vzdálený repozitář již existující lokální repozitář s určitou historií změn, volba ***Initialize repository with a README*** **nesmí být zaškrtnutá!** Způsobilo by to konfliktní historii těchto dvou repozitářů a nebylo by možné je na sebe navázat.



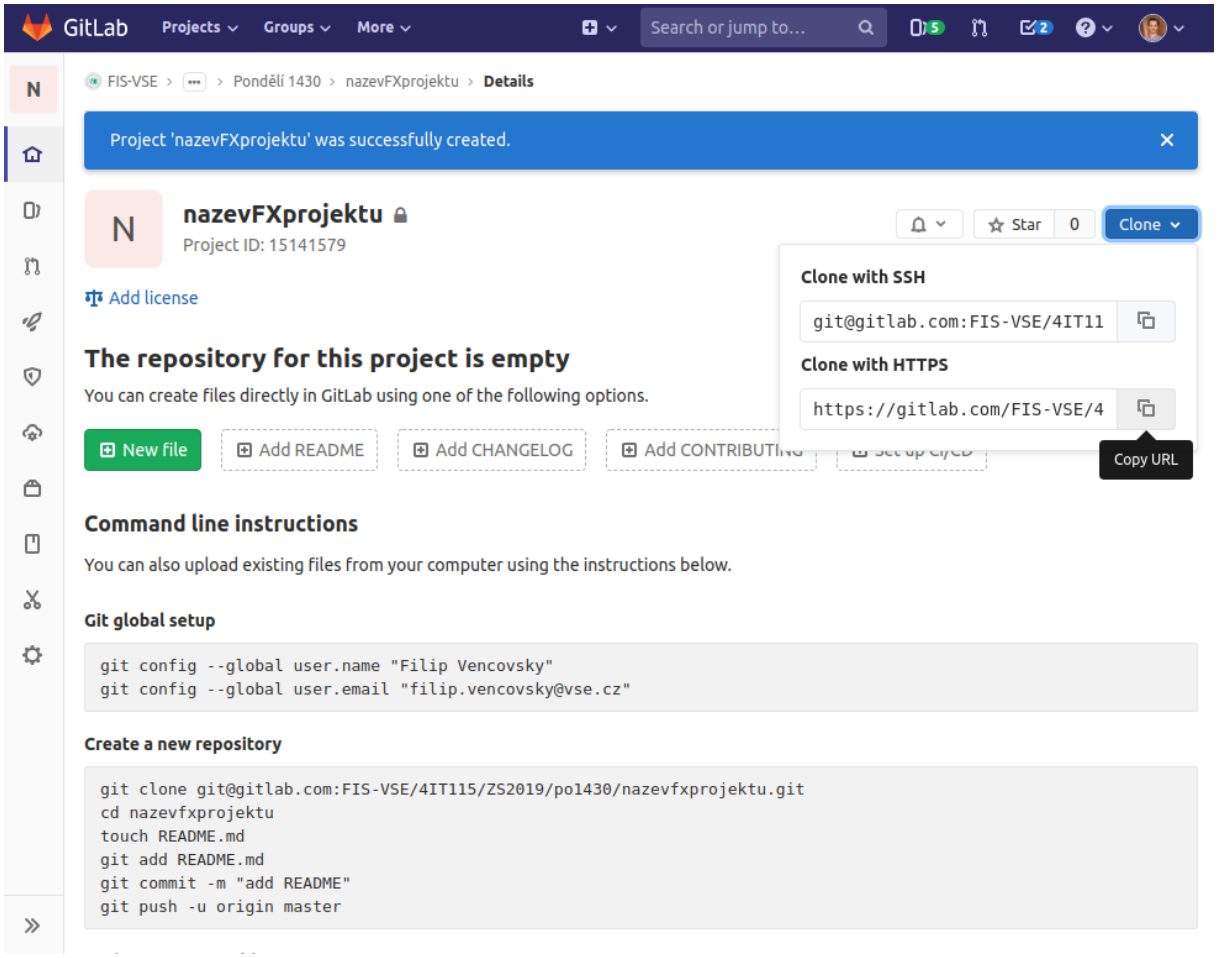
Obrázek 8.12: Založení nového projektu na GitLab.com

8.4 Propojení lokálního a vzdáleného repozitáře

Pro propojení lokálního a vzdáleného repozitáře je nutné zjistit URL vzdáleného repozitáře. Po otevření prázdného projektu na GitLab.com klikněte na modré tlačítko *Clone* a zkopírujte adresu kliknutím na symbol kopie u jednoho ze způsobů klonování *SSH* nebo *HTTPS*.

Podrobnosti o protokolech najdete v kapitole Klonování existujícího projektu.

Při kopírování URL dejte pozor, aby adresa končila na *.git*.



Obrázek 8.13: Kopírování URL nového projektu na Gitlab.com

8.4.1 Propojení repozitářů v příkazové řádce

V příkazové řádce přidáte vzdálený repozitář jednoduše příkazem:

```
git remote add názevRepozitáře URLrepozitáře
```

Jelikož se jedná o první vzdálený repozitář, použijeme obvyklý název `origin`. URL repozitáře zkopírujeme z úvodní stránky našeho projektu na GitLab.com, viz předchozí postup.

Pro SSH například:

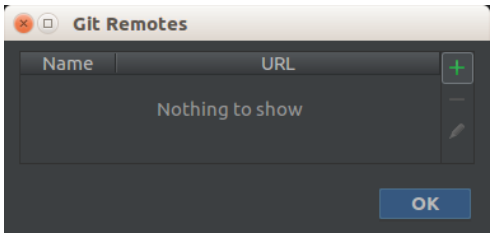
```
git remote add origin git@gitlab.com:priklad/nazevfxprojektu.git
```

Pro HTTPS například:

```
git remote add origin https://gitlab.com/priklad/nazevfxprojektu.git
```

8.4.2 Propojení repozitářů v IntelliJ IDEA

V IntelliJ IDEA se napojení na vzdálený repozitář provede v dialogu *Git Remotes*, který se otevře z hlavní nabídky *VCS* > *Git* > *Remotes...*.

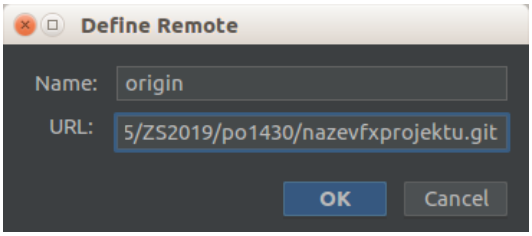


Obrázek 8.14: Dialog pro přidání vzdáleného repozitáře v IntelliJ IDEA

První vzdálený repozitář se přidá stisknutím tlačítka se symbolem plus v pravé horní části dialogu.

Do pole *Name* vyplňte název vzdáleného repozitáře. Jelikož se v tomto případě jedná o první vzdálený repozitář, je zvykem pojmenovat ho *origin*.

Do pole *URL* vložte zkopírovanou adresu vzdáleného repozitáře.



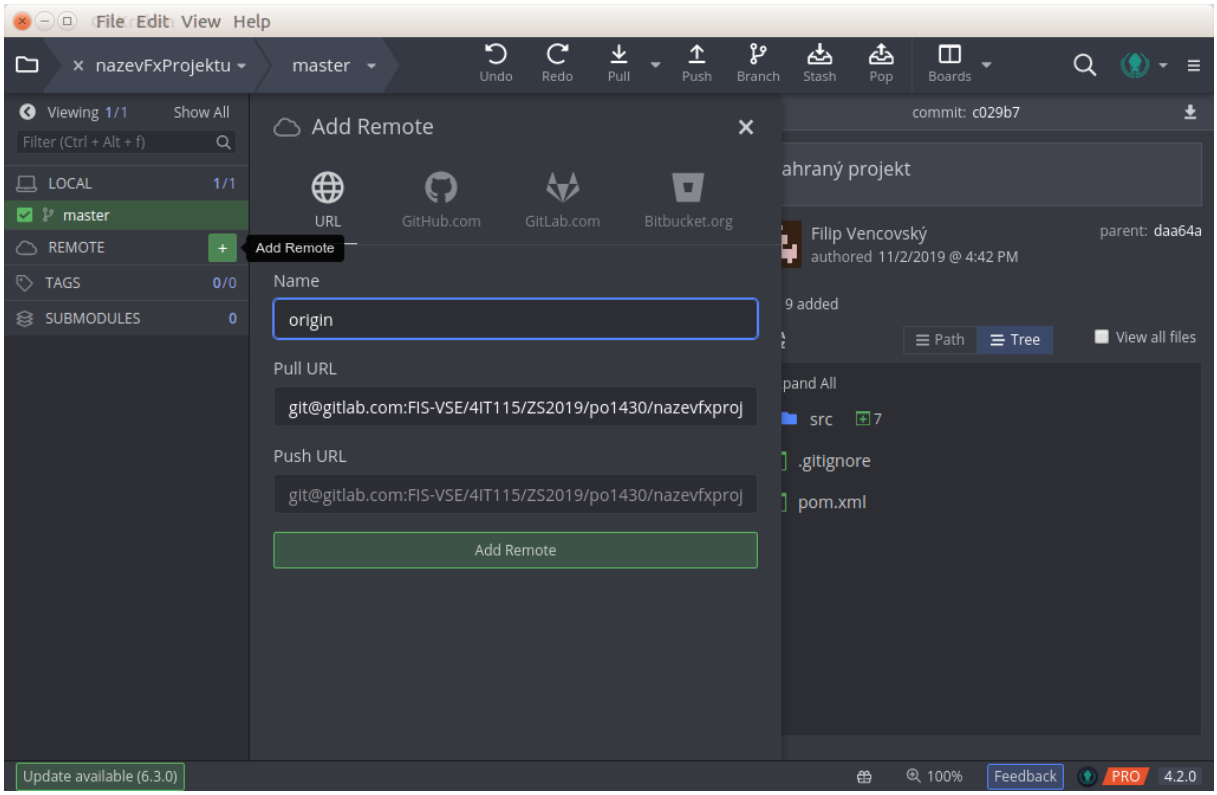
Obrázek 8.15: Napojení na vzdálený repozitář v IntelliJ IDEA

8.4.3 Propojení repozitářů v GitKraken

V GitKraken se vzdálený repozitář přidá stisknutím symbolu plus, který se objeví po najetí myši nad položku *REMOTE* v levém panelu.

Po otevření dialogu *Add Remote* zvolte záložku *URL* a vložte zkopírovanou adresu vzdáleného repozitáře do pole *Pull URL* a *Push URL*.

Jelikož se jedná o první vzdálený repozitář, je obvyklé pojmenovat ho *origin* (pole *Name*).



Obrázek 8.16: Napojení na vzdálený repozitář v GitKraken

8.5 Odeslání lokálních změn na vzdálený repozitář

Po propojení lokálního a vzdáleného repozitáře je možné odeslat dosavadní zaznamenané revize z lokálního repozitáře na vzdálený příkazem *push*.

V příkazové řádce se *push* provede jednoduše zadáním `git push`.

V GitKraken se příkaz provede kliknutím na tlačítko *Push* v hlavní nástrojové liště v horní části okna.

V IntelliJ se příkaz odesílá z dialogového okna *Push Commits*, které otevřete z hlavní nabídky *VCS* > *Git...* > *Push...*

8.6 Alternativní postup sdílení projektu

Alternativně je možné naklonovat prázdný vzdálený repozitář a soubory z nenasdíleného projektu přesunout do nově naklonovaného. Tento postup ovšem přináší následující rizika:

- Projekt bude muset nést jiný název nebo bude muset být vytvořený v jiném umístění na disku, aby nedošlo ke konfliktu v IDE.
- Při manuálním kopírování je riziko, že se nepodaří přenést všechny soubory nebo že budou vznikat konflikty v nastavení projektů.

8.7 Časté chyby při sdílení projektu

Při sdílení projektu se můžete setkat s následujícími problémy:

- Při prvním přihlášení na vzdálený repozitář přes HTTPS se může stát, že zadáte špatné heslo. Pokus o připojení ke vzdálenému repozitáři může stále vracet chybu, protože Git klient se na nové heslo i při nezdařených pokusech už nikdy nezeptá. Problém komplikuje také to, že heslo mohlo být uloženo různým způsobem: *IntelliJ IDEA ukládá heslo dle nastavení, viz heslo Passwords v dokumentaci.* Čistý Git používá *credential helper*, který je možné resetovat příkazem `git config --system --unset credential.helper`, viz gitcredentials v dokumentaci. *Ve výjimečných případech může být heslo uložené na úrovni operačního systému, např.* Windows Credentials*.
- Inicializace vzdáleného repozitáře při vytváření způsobuje konflikt historie při pokusu o propojení s lokálním repozitářem. Při spojování se ujistěte, že jeden z repozitářů je prázdný.

Kapitola 9

Klonování existujícího projektu

Klonování projektu je postup, při kterém se vytvoří lokální úložiště jako obraz vzdáleného.

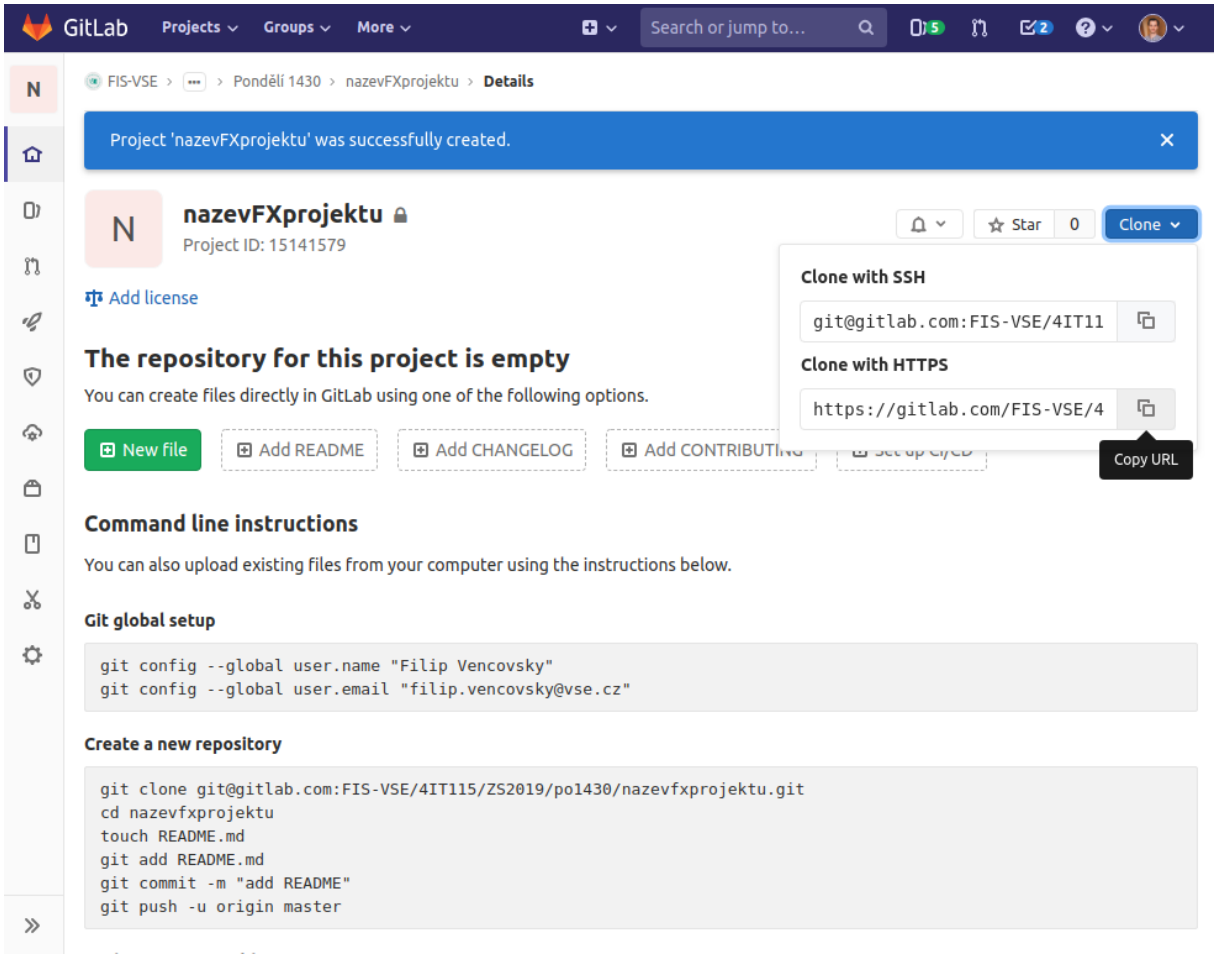
Klonování existujícího projektu je o poznání snazší než sdílení projektu. Stejně jako u sdílení je možné použít přímo git, IDE nebo GUI klient obsluhující git (např. *GitKraken*).

Všechny způsoby používají příkaz `git clone`. Ten vždy vytvoří nové lokální úložiště a zrcadlí jeho obsah, včetně historie. Stažený projekt je následně potřeba otevřít v IDE, které načte projektovou konfiguraci z objektového modelu *Maven* (`pom.xml`) nebo jiných konfiguračních souborů, které jsou k tomu účelu k dispozici (*Gradle*, nastavení IDE).

Pro klonování potřebujete znát URL vzdáleného repozitáře. V případě GitLab ho najdete na úvodní stránce projektu pod modrým tlačítkem *Clone*. URL závisí na protokolu přenosu souborů. Je možné vybrat buď *SSH*, nebo *HTTPS*.

Doporučený způsob je *SSH*, vyžaduje ale vygenerování a nahrání veřejného klíče na GitLab.com. Metoda je bezpečnější a při práci se vzdáleným repozitářem není potřeba zadávat heslo. Při zvolení *HTTPS* je heslo nutné, ale většina programů jej umí uložit. Při použití *HTTPS* dejte pozor na uložení špatně zadaného hesla, viz podkapitola Časté chyby.

Všimněte si, že kopírovaná adresa repozitáře končí *.git*, na rozdíl od URL stránek projektu, které je až na koncovku stejné.

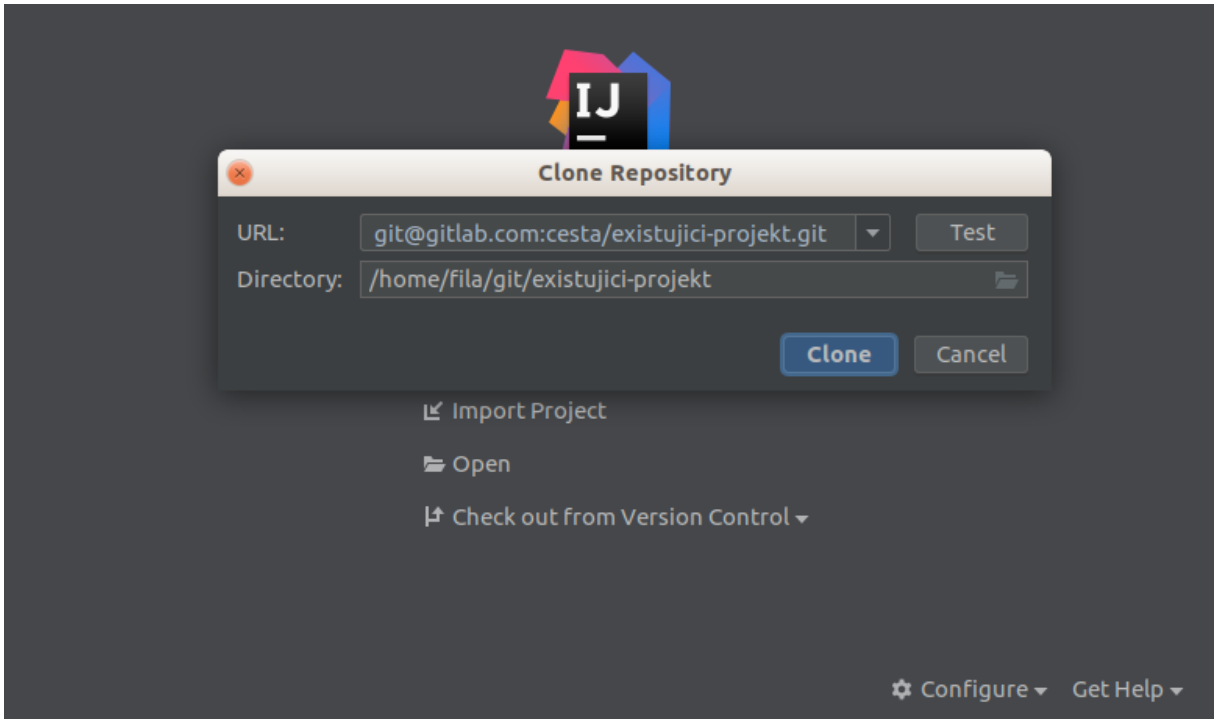


Obrázek 9.1: Kopírování URL nového projektu na Gitlab.com

9.1 Klonování pomocí IDE

Klonování pomocí IDE má výhodu, že výše popsaný postup proběhne v jednom kroku.

Na úvodní obrazovce IntelliJ IDEA zvolte položku *Check out from Version Control*. Objeví se dialog *Clone Repository*. Do pole *URL* vložte zkopírované URL. V poli *Directory* se vygeneruje adresář pro umístění projektu. Cestu zkontrolujte a klonování dokončete tlačítkem *Clone*.



Obrázek 9.2: Klonování existujícího projektu v IntelliJ IDEA

Kapitola 10

Architektura JavaFX projektu

Kapitola popisuje architektonické volby pro novou JavaFX aplikaci a představuje minimální architektonický návrh nutný pro vykreslení okna a přípravu na grafické prvky a ovládání jejich událostí.

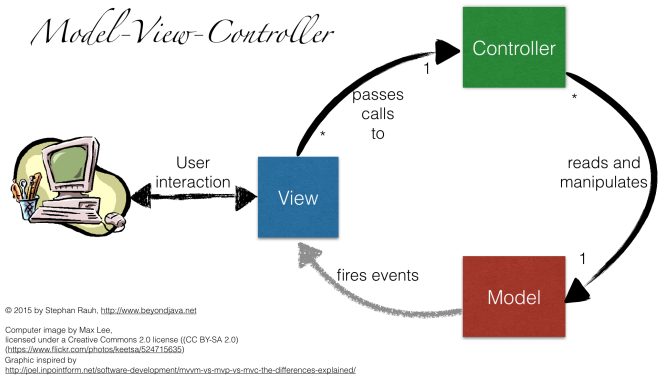
Při návrhu architektury je výhodné aplikaci rozdělit do menších celků, které usnadní její správu a umožní lepší spolupráci v týmu vývojářů. Způsobů, jak aplikaci rozdělit, je mnoho. Obecně se jako dobrá praxe vžilo minimálně **oddělení uživatelského rozhraní, byznys logiky a dat**, které reprezentují architektonické vzory MVx: Model-View-Controller (MVC), Model-View-Presenter (MVP), Model-View-ViewModel (MVVM). Jaký návrhový vzor a jaká implementace vzoru je pro JavaFX aplikace nejvhodnější?

Kritéria pro volbu architektury:

- srozumitelnost kódu,
- snadné testování,
- snazší rozdělení týmových rolí,
- minimalizace duplicitního kódu (boilerplate).

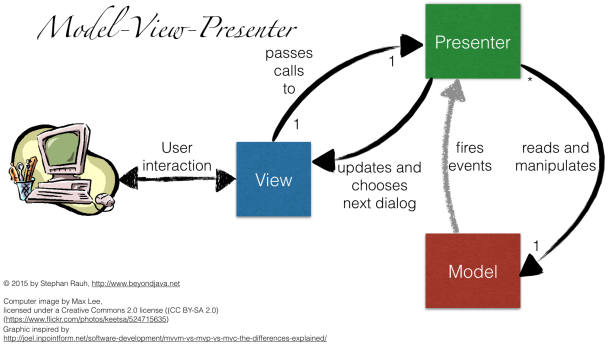
Implementace architektury se liší zejména tím, jak jsou jednotlivé části vzájemně propojené a jak spolupracují. Rozdíly velmi dobře shrnuje článek Model-View-Whatever na BeyondJava.

Princip návrhového vzoru **Model-View-Controller** (MVC) spočívá v tom, že *view* reaguje na vstupy od uživatele a posílá o nich zprávy kontroleru (*controller*), který upravuje *model*. Model potom posílá informace o změnách *view*.



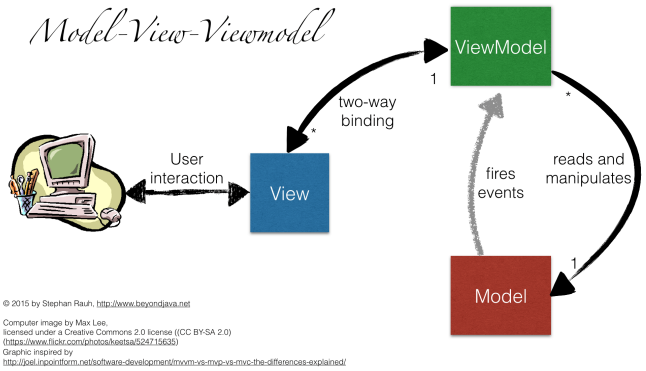
Obrázek 10.1: Model-View-Controller (zdroj: Stephan Rauth)

Model-View-Presenter (MVP) provázanost snižuje a namísto kontroleru, který by jen zachytával vstupy uživatele, představuje prezentér (*presenter*). Prezentér funguje jako mezivrstva. Je odpovědný za komunikaci mezi *view* a modelem.



Obrázek 10.2: Model-View-Presenter (zdroj: Stephan Rauth)

Model-View-ViewModel (MVVM) využívá v komunikaci mezi view a prostředníkem *viewmodel* princip *data binding*, který se hojně používá při vývoji aplikací pro Android. Data binding si lze zjednodušeně představit jako propojení na bázi synchronizace.



Obrázek 10.3: Model-View-ViewModel (zdroj: Stephan Rauth)

10.1 Volba implementace MVx

Jednou z výhod, kterou má JavaFX oproti starším grafickým knihovnám, je možnost použití **deklarativního zápisu** grafického rozhraní formou FXML.

Výhody použití deklarativního zápisu:

- možnost využít aplikaci pro usnadnění návrhu (např. Scene Builder),
- rychlejší tvorba rozhraní pro účely uživatelského testování,
- návrh UI je možné uskutečnit bez znalosti programování.

Při tvorbě JavaFX aplikace za pomoci FXML by se mohlo zdát, že už implementace směřuje určitým směrem. JavaFX knihovna dokáže propojit jeden FXML soubor s právě jedním `fx:controller`. Ten není definován jako potomek žádné knihovní třídy, takže je na vývojáři, jak třídu deklaruje, ale správně by měl být potomkem kořenového prvku FXML (například *GridPane*). `fx:controller` má za cíl řešit přístup k jednotlivým ovládacím prvkům a řešit reakce na vstupy uživatele.

Takto definovaný `fx:controller` lze chápat jako *presenter* z MVP nebo jako aktivní část *view*, normálně pasivního view deklarovaného v FXML.

`fx:controller` nemůže být plnohodnotným MVC kontrolerem proto, že je přímo svázán s konkrétním FXML a nedokáže tak vyřešit přepínání jednotlivých *view*.

Vnitřní struktura JavaFX s FXML tedy nebrání různým implementacím MVx. Například velmi působivá je implementace Model-View-ViewModel ve frameworku MvvmFX.

10.2 Minimální architektonický návrh JavaFX aplikace

Pokud bychom pro návrh architektury pro JavaFX aplikaci s použitím FXML chtěli zvolit co nejjednodušší cestu, vytvoříme:

- byznys třídy reprezentující logiku aplikace,
- spouštěcí třídu, která je potomkem *Application*,
- deklarativní zápis UI ve formě FXML,
- ovladač událostí FXML (`fx:controller`).

V tomto případě je architektura nejblíže MVP. Uživatel interaguje s grafickými ovládacími prvky (*view*), které spouští události v `fx:controller` (*presenter*). Prezentérů může být více, ale mají odkaz na jedno rozhraní byznys tříd (*model*), se kterým mohou manipulovat.

Podstatné je, že jsme **oddělili byznys třídy a třídy UI**. Nezávislost těchto komponent je předpokladem pro dobrou architekturu a vyhovuje širší definici MVC.

10.3 Reakce na události z modelu

Vhodné doplnění takovéto architektury je **návrhový vzor observer**, který vyřeší situace, kdy potřebujeme reagovat na události vznikající v modelu. Pokud bychom návrhový vzor observer nepoužili, nebylo by možné reagovat na události z modelu v UI jiným způsobem než použitím odkazu na třídy UI, což je v rozporu s principem oddělení byznys tříd a tříd UI.

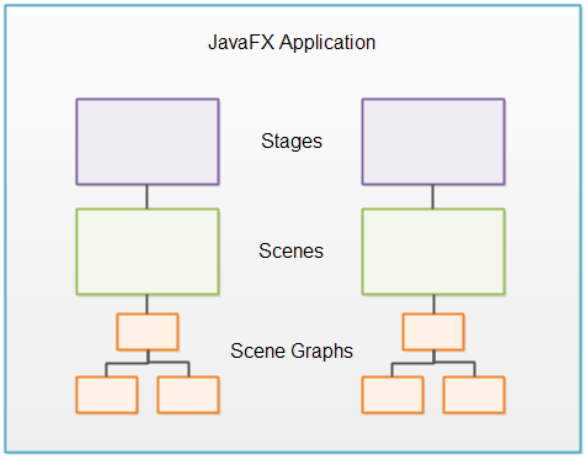
Návrhový vzor observer funguje na principu „odběru novinek“. Ten, kdo je producentem událostí (*observable*), upozorňuje své odběratele (*observer*), že událost nastala. Producentem událostí může být rozhraní byznys tříd nebo jakákoli třída, kterou rozhraní dokáže předat. Odběratelem událostí může být určitý prezentér (`fx:controller`) nebo jiná třída, která je odpovědná za UI.

Návrhový vzor je názorně popsaný v článku Observer Design Pattern in Java.

10.4 Okna aplikace a jejich obsah

Okno aplikace je tvořeno objektem třídy *Stage*. Na objektu *Stage* se, podobně jako na každém jiném pódiu, odehrávají různé scény. Objekt třídy *Scene* je vyjádřením konkrétní scény. Scéna potom obsahuje konkrétní JavaFX komponenty (*scene graph*), jako např. GridPane, Button, TextField, HBox a další. Komponenty mají hierarchickou strukturu a všechny dědí z třídy *Node*, česky uzel. Uzly, které umožňují nést další uzly, jako např. GridPane či HBox, dědí z třídy *Parent*. Rodičovský uzel (nebo také kořen či *root*) je na vrcholu hierarchie komponent tvořících obsah scény. Každá komponenta uživatelského rozhraní je v něm obsažena. Kořen je zapotřebí ke konstrukci nové scény.

```
Parent root = new GridPane();
Scene scene = new Scene(root);
Stage stage = new Stage(scene);
```



Obrázek 10.4: Schéma prvků JavaFX (zdroj: Jenkov.com)

10.4.1 Zobrazení okna s obsahem

Grafické rozhraní JavaFX aplikace se spustí pomocí statické metody `launch()`, kterou obsahuje každý potomek třídy *Application*.

Vhodné místo je například metoda `main()`.

```
public class Main extends Application {
  public static void main(String[] args) {
    launch(args);
  }
}
```

Kód pro sestavení a zobrazení scény je třeba umístit do metody `start()`, která je povinná u tříd, které dědí z třídy *Application*.

Metoda `start()` má mezi parametry již předem vytvořený objekt třídy *Stage*. Takto předanému objektu se jen nastaví nová scéna.

```
@Override
public void start(Stage primaryStage) throws Exception {
  Parent root = new GridPane();
  Scene scene = new Scene(root);

  // nastavení nové scény pro primární stage
  primaryStage.setScene(scene);
```

```
  //zobrazení okna
  primaryStage.show();
}
```

10.4.2 Zobrazení FXML

V případě FXML je nutné použít třídu *FXMLLoader*, která dokáže soubor načíst a vrátit odkaz na kořen (*root*). K tomu slouží statická metoda `load(URL location)`. Parametrem metody je objekt třídy *URL*, který směřuje na umístění FXML souboru.

V případě Maven projektu je možné využít standardní lokace pro zdroje a soubor umístit do adresáře `src/main/resources/` voláním `getClass().getResource("/fxmlsoubor.fxml")`.

```
@Override
public void start(Stage primaryStage) throws Exception {
  // získání URL souboru
  URL fxmlSoubor = getClass().getResource("/fxmlsoubor.fxml");

  // načetení FXML souboru
  Parent root = FXMLLoader.load(fxmlSoubor)

  Scene scene = new Scene(root);
  primaryStage.setScene(scene);
  primaryStage.show();
}
```

10.4.3 Časté chyby

- FXML soubor není umístěný v předkonfigurovaném (*Maven*) adresáři pro zdroje `src/main/resources/`.
- Chybí lomítko u cesty k FXML souboru. Lomítko označuje, že jde o kořen adresáře určeného pro zdroje (*Maven*).
- V některých případech při použití Java SDK verze 11 je nutné nakonfigurovat správně moduly aplikace v souboru `module-info.java`, viz založení nového projektu pro Javu verze 11.
- Aplikace se nezobrazí, protože chybí volání metody `show()`.
- FXML obsahuje chyby v syntaxi, a proto soubor nelze načíst. Je dobré vždy otestovat, zda funguje prázdné FXML jen s kořenovým prvkem, např. GridPane.

Kapitola 11

Sestavení projektu

Ať je výsledná forma spustitelný archiv *jar*, webová aplikace *war*, nebo instalovatelný program pro operační systémy Windows, Linux nebo Mac, sestavení aplikace je významná část každého projektu.

Spustitelný archiv *jar* (*Java archive*) je založený na formátu *zip*. Nese zkompilované Java třídy ve struktuře odpovídající balíčkům, zdroje jako obrázky nebo konfigurační soubory a metainformace. Archiv je spustitelný právě díky metainformacím, které obsahují cestu ke spustitelné třídě. Archiv může obsahovat také kontrolu integrity a digitální podpis. Více informací se dozvíte v dokumentaci na stránce JAR File Specification.

Kvůli velkému důrazu na automatizaci, který se na vývoj softwaru klade, je optimálním způsobem sestavení takový způsob, který je nezávislý na použitém vývojovém prostředí. Tím se dostáváme k výhodám založení projektu pomocí Maven nebo Gradle, kde je sestavení možné jedním voláním v příkazové řádce.

Maven obsahuje standardně konfiguraci, která dokáže spustitelný archiv sestavit. Za použití pluginů nabízí široké množství přednastavených procedur pro sestavení do různých forem.

Kromě sestavení pomocí Maven se kapitola věnuje také sestavení programem jar a v rámci IDE.

11.1 Maven projekt

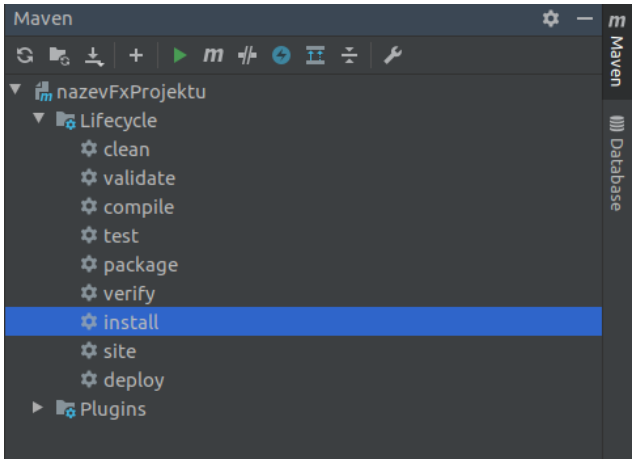
Maven projekt je velice jednoduché sestavit. K tomu účelu se používá sled příkazů:

```
clean install
```

Příkaz **clean** kompletně smaže složku *target*, aby byl projekt sestaven bezpečně bez reziduí z minulých sestavení. Příkaz **install** se postará o vše ostatní: zkompiluje kód, překopíruje zdroje, provede testy a vše zabalí do archivu.

Více informací se dočtete v Introduction to the Build Lifecycle v dokumentaci.

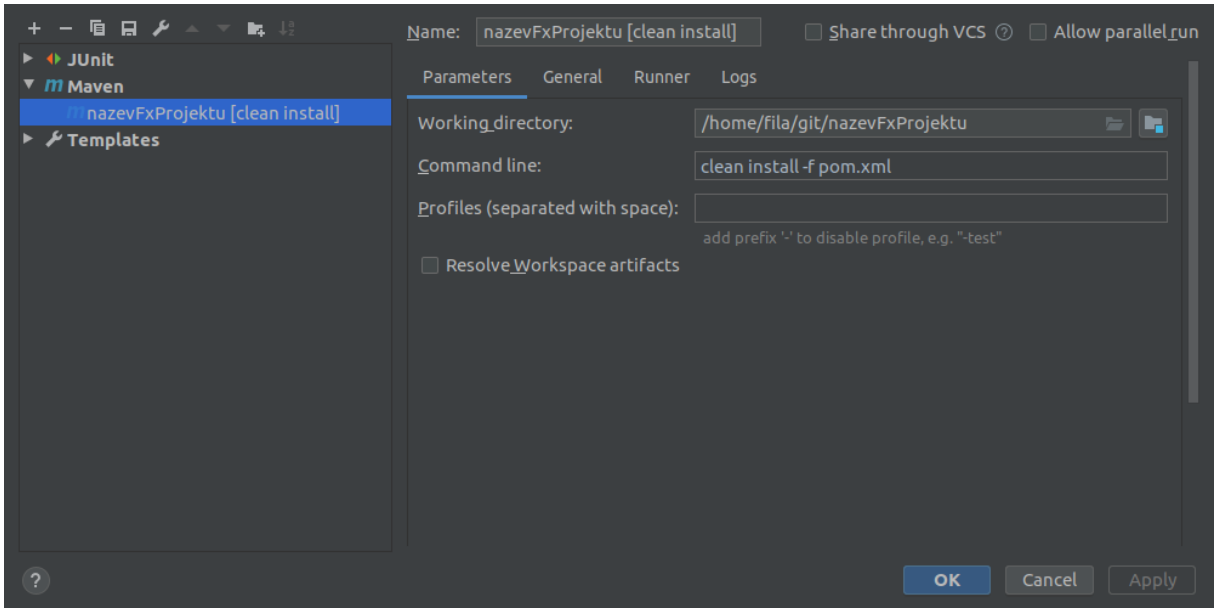
V IDEA můžete jednotlivé příkazy zavolat dvojklikem na příkaz v pohledu přes fáze životního cyklu (*Lifecycle*) v Maven panelu.



Obrázek 11.1: Volání příkazů clean a install

Případně je možné nastavit `clean install` jako spustitelnou konfiguraci. Spustitelná konfigurace se nastavuje volbou *Run* > *Run...* z hlavního menu. V upřesňujícím dialogu, který se následně otevře, zvolte *Edit Configurations...* Tím se spustí dialog spustitelných konfigurací.

Stisknutím tlačítka se symbolem plus se otevře seznam šablon, mezi kterými je *Maven*. Všechny parametry je možné ponechat a vyplnit jen příkaz (*Command line*) `clean install -f pom.xml`. V příkazu je navíc parametr `-f`, který upřesňuje soubor, ve kterém je projektový objektový model (*pom.xml*).



Obrázek 11.2: Konfigurace volální clean install

To, že součástí sestavení je automatické **provedení jednotkových testů**, je značná výhoda. Při selhání testů se totiž výsledný archiv nevytvoří a nemůže se tak stát, že bude existovat „nefungující“ verze programu.

Plugin pro spustitelný archiv *jar* je už předdefinovaný v archetypu, který používáme. Jediné, na co je třeba si dávat pozor, je **cesta ke spustitelné třídě** a **připojení závislostí**.

Konfigurace pluginu se v objektovém modelu (*POM*) umísťuje vedle ostatních konfigurací pluginů do elementu `<plugins>`, který je součástí `<pluginManagement>` a `<build>` na nejvyšší úrovni.

Cesta ke spustitelné třídě je obsahem elementu `<mainClass>` a musí odpovídat plné cestě přes balíčky až po název třídy. Při sestavování se zapíše v archivu do souboru `/META-INF/MANIFEST.MF`.

V situaci, kdy projekt používá **závislosti**, se mohou vyskytnout problémy s přítomností těchto závislostí v archivu. Pro tuto situaci se nabízí dvě řešení:

1. přidat závislosti do spustitelného archivu,
2. zkopírovat závislosti do cílové složky vedle spustitelného archivu.

Pokud to situace umožňuje, je bezpečnější první způsob: vložení závislostí do archivu. Vhodné to nemusí být v případech, kdy to neumožňují licenční pravidla, nebo třeba při větší velikosti připojovaných archivů. Například natrénované modely pro rozpoznávání obrazu nebo textu mohou nabývat velikosti v gigabytech a mohou ztížit manipulaci s programem a jeho aktualizace.

Variantám tvorby archivu se věnuje článek na Baeldung.

11.1.1 Zahrnutí závislostí do archivu

Závislosti se do spustitelného archivu přidají za použití The Apache Maven Assembly Plugin s následující konfigurací v projektovém objektovém modelu (*POM*). Plugin je identifikovaný pomocí *groupId* `org.apache.maven.plugins` a *artefactId* `maven-assembly-plugin`.

```
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-assembly-plugin</artifactId>
    <executions>
        <execution>
            <phase>package</phase>
            <goals>
                <goal>single</goal>
            </goals>
            <configuration>
                <archive>
                <manifest>
                    <mainClass>
                        cesta.SpustitelnaTrida
                    </mainClass>
                </manifest>
                </archive>
                <descriptorRefs>
                    <descriptorRef>jar-with-dependencies</descriptorRef>
                </descriptorRefs>
            </configuration>
        </execution>
    </executions>
</plugin>
```

11.1.2 Ponechání závislostí mimo archiv

Pokud nechceme závislosti zahrnovat do archivu, můžeme použít standardní jar plugin, ale bude nutné závislosti zkopírovat do složky *target*.

Ke kopírování lze použít `maven-dependency-plugin`. Při konfiguraci je nutné věnovat pozornost elementu `<outputDirectory>`, který určuje, do které složky v rámci *target* se mají závislosti překopírovat.

```
<!-- standardní jar plugin -->
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-jar-plugin</artifactId>
    <configuration>
        <archive>
            <manifest>
                <addClasspath>true</addClasspath>
                <classpathPrefix>libs/</classpathPrefix>
                <mainClass>
                    cesta.SpustitelnaTrida
                </mainClass>
            </manifest>
        </archive>
    </configuration>
</plugin>

<!-- kopírování závislostí -->
<plugin>
    <groupId>org.apache.maven.plugins</groupId>
    <artifactId>maven-dependency-plugin</artifactId>
    <executions>
        <execution>
            <id>copy-dependencies</id>
            <phase>prepare-package</phase>
            <goals>
                <goal>copy-dependencies</goal>
            </goals>
            <configuration>
                <outputDirectory>
                    ${project.build.directory}/libs
                </outputDirectory>
            </configuration>
        </execution>
    </executions>
</plugin>
```

Při ponechání závislostí mimo archiv vzniká **riziko neúplného přenesení aplikace**. Pro snížení rizika je možné aplikaci sestavit jako **instalovatelnou**. K tomu je možné použít např. plugin IzPack.

11.2 Alternativní způsoby sestavení

11.2.1 Program jar

Pro sestavení archivu pomocí programu *jar* se používá příkaz definovaný jako

```
jar [OPTION...] [ [--release VERSION] [-C dir] files] ...
```

Spustitelný archiv se vytvoří například následujícím příkazem. Přepínač `--create` nastaví, že archiv se bude vytvářet, nikoli aktualizovat. Parametr `--file` určí výsledný soubor. Parametr `--main-class` nastaví cestu ke spustitelné třídě. Parametr `-C` stanoví, ve kterém adresáři se nachází soubory, které se mají zabalit. Následuje tečka `.`, která reprezentuje všechny soubory v tomto adresáři.

```
jar --create --file target/nazevFxProjektu-1.0.0.jar
```

```
--main-class cesta.SpustitelnaTrida -C target/classes/ .
```

Toto řešení nezahrne do archivu Maven závislosti a před sestavením se neprovedou testy. Proto je doporučené archiv vytvářet pomocí programu *Maven*.

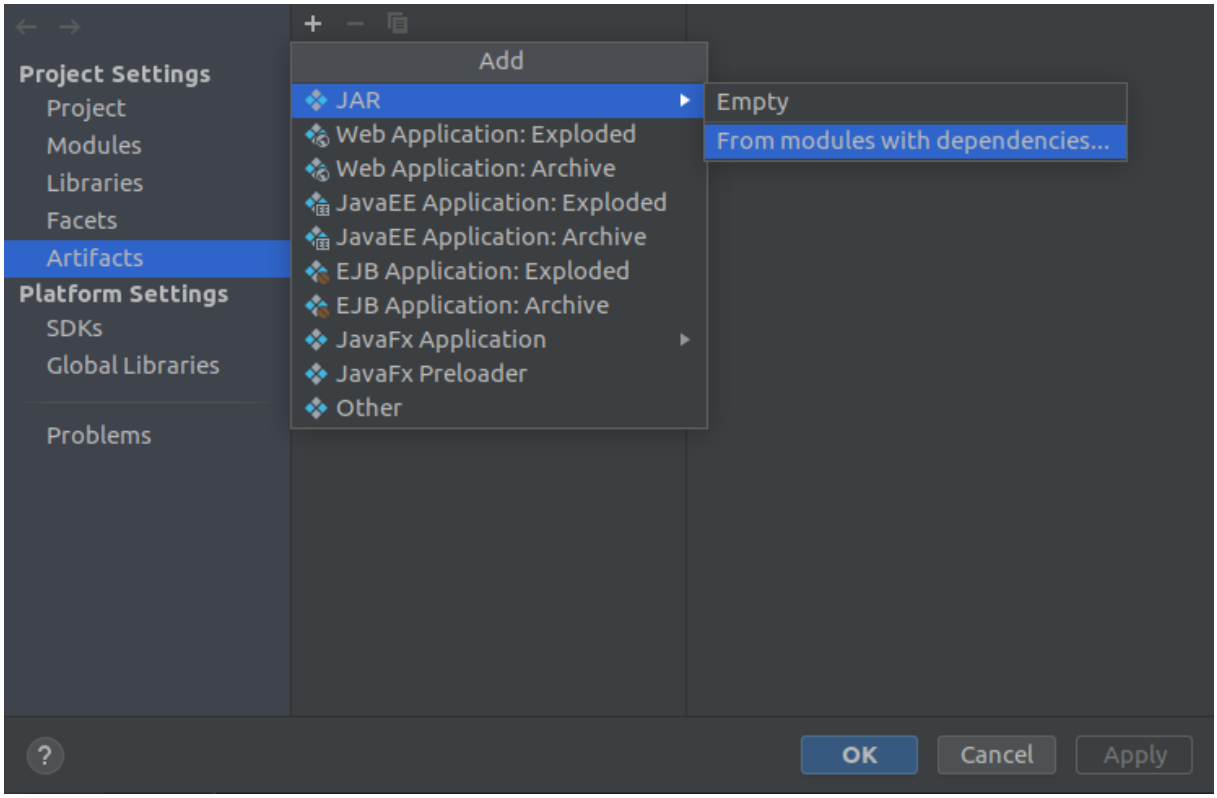
Více informací o programu *jar* najdete v dokumentaci.

11.2.2 Sestavení v IDE

Alternativou k programu *jar* je sestavení archivu pomocí vývojového prostředí. Tím, že projekt ve vývojovém prostředí už obsahuje určitá nastavení, např. závislosti na knihovnách, je proces sestavení uživatelsky méně náročný. Následující příklad popisuje práci na sestavení v IntelliJ IDEA.

Předtím než je možné v IntelliJ IDEA spustit samotné sestavení, je třeba nakonfigurovat výstupní artefakt archivu. To je možné v dialogu *Artifacts*, který je dostupný po zvolení volby *File > Project Structure...*

Artefakt se přidává stiskem tlačítka se symbolem plus a vybráním příslušného typu artefaktu, v našem případě *JAR*.

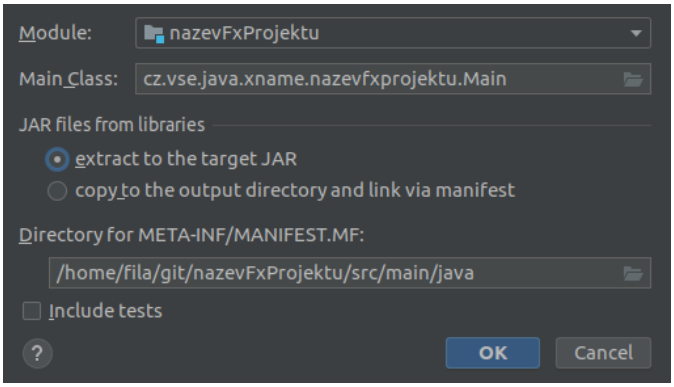


Obrázek 11.3: Dialog k artefaktům v nastavení projektu

Otevře se dialog s nastavením archivu. V něm je třeba vybrat modul projektu (*Module*). V našem případě jednomodulového projektu je jediná možnost, a to zvolit spustitelnou třídu s její plnou cestou (*Main Class*) a vybrat, zda do archivu mají být zahrnuty závislosti.

Adresář *META-INF* je třeba umístit do adresáře *java*, který obsahuje strukturu adresářů odpovídající balíčkům. Toto místo se obvykle zvolí automaticky bez problému.

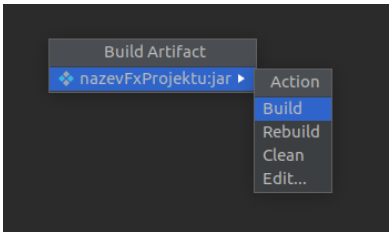
Poslední volba se týká zahrnutí testů do archivu (*Include tests*). Nejde ovšem o jejich provedení při sestavování jako ověření kvality, které se provádí v případě *Maven install*, ale skutečně o zahrnutí testovacích tříd do archivu. Z toho důvodu můžeme ponechat volbu nezaškrtnutou.



Obrázek 11.4: Dialog přidání nového jar archivu jako artefaktu

Po potvrzení dialogu se, kromě konfiguračních položek v *.idea*, vytvoří v adresáři *src/main/java/META-INF* soubor *MANIFEST.MF*, ve kterém je uložená cesta ke spustitelné třídě.

Archiv se potom sestaví volbou *Build > Build Artifacts...* a výběrem artefaktu a příslušné akce. Možné akce jsou sestavení (*Build*),



Obrázek 11.5: Dialog k sestavení archivu

Tento postup zahrne do výsledného archivu závislosti, ale na rozdíl od sestavení archivu pomocí *Maven* nebo *Gradle* nelze použít pro automatizaci, například v integračním skriptu na GitLab.com nebo Jenkins serveru.

Oficiální návod najdete na stránce Working with Artifacts.

11.3 Spuštění archivu

Hotový archiv *jar* se spouští nejlépe v **příkazovém řádku** pomocí `java -jar nzevFxProjektu-1.0-SN`

Pokud spouštění dělá problémy, ujistěte se, že používáte kompatibilní verzi Java s tou, na kterou byl projekt sestaven, příkazem `java -version`.

Kapitola 12

Týmová práce při vývoji softwaru

Vývoj softwaru v praxi nebývá individuální záležitost. Práce v týmu klade vysoké nároky na koordinaci práce vývojářů. Podcenění těchto aktivit může mít velký dopad na kvalitu nebo dobu trvání projektu. Navíc se v současné době stále častěji prosazuje tendence dodávat software v krátkých intervalech.

Organizaci práce na projektu a koordinaci týmu se věnuje mnoho metodik vývoje softwaru. Příkladem metodik jsou Scum, Kanban, OpenUp a z něho odvozená MMSP. Podrobnosti o metodikách jsou popsány v knize

> BUCHALCEVOVÁ, Alena. Zlepšování procesů při budování informačních systémů. Vydání první. Praha: Oeconomica, nakladatelství VŠE, 2018. 227 stran. ISBN 978-80-245-2235-7.

Integrovaná vývojová prostředí (IDE) pro aktivity vývoje v týmu nabízí jen minimální podporu. Následující seznam je příkladem typů nástrojů, které určitou míru podpory týmového vývoje nabízí:

- issue tracking,
- kanban boards,
- větvení při správě verzí,
- kontinuální integrace a nasazení (*CI/CD*),
- komunikační nástroje.

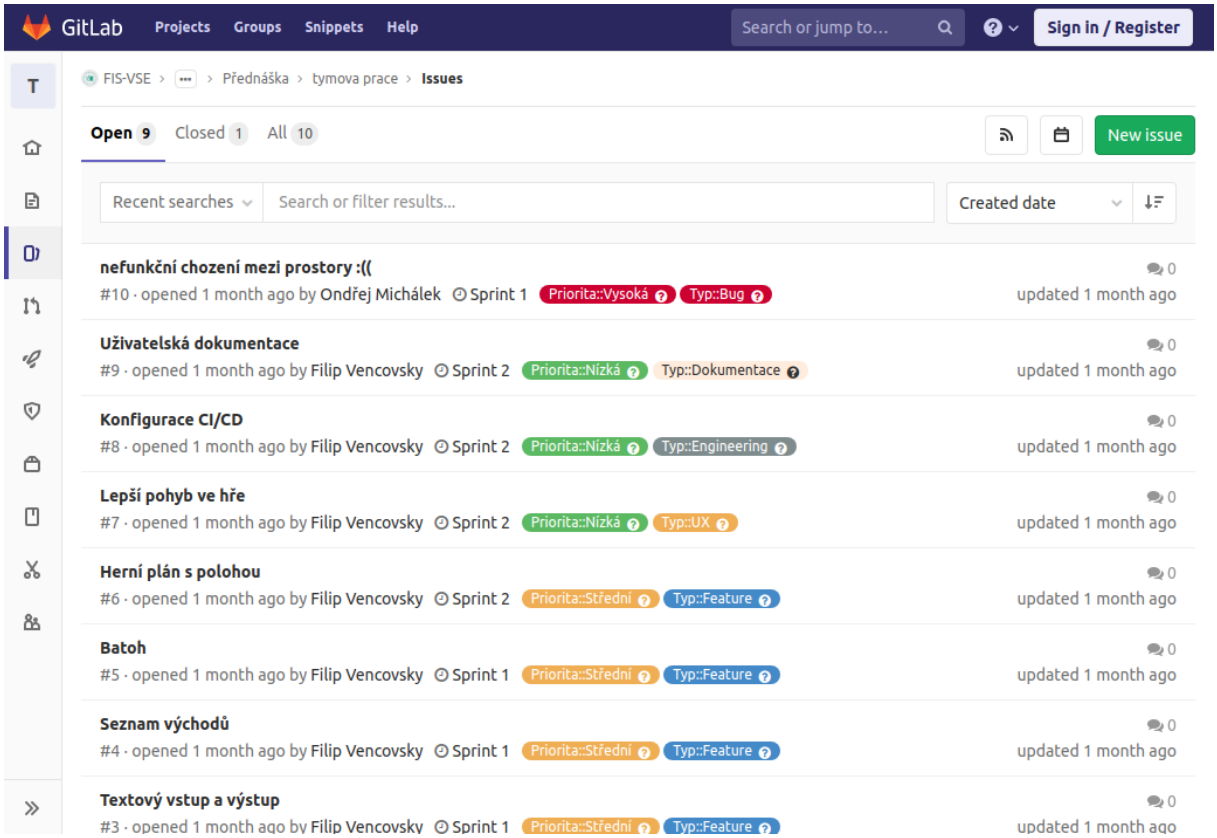
Kapitola popisuje nástroje, které podporují týmovou práci a jsou dostupné na platformě GitLab.

12.1 Issue tracking

Termín *issue* pokrývá v kontextu vývoje softwaru širokou řadu užití. Může se jednat o

- **otázky** a **problémy**, které je potřeba během vývoje řešit;
- **chyby** v aplikaci identifikované v provozu nebo při testování;
- **požadavky** na vývoj produktu;
- nové **nápady** k diskuzi.

Každý issue má svého původce, osobu, která např. identifikuje chybu nebo nový nápad. Původce zakládá issue do systému GitLab vložením jeho **názvu** a **popisu**.



Obrázek 12.1: Seznam issues na GitLab.com

K issue je následně potřeba přiřadit **zodpovědnou osobu**. Pravidla pro určování zodpovědných osob je třeba určit před začátkem projektu. Zodpovědnou osobu může určit např. vedoucí týmu nebo se osoba může přiřadit k issue sama. Se zodpovědnou osobou je vhodné vyplnit i **termín pro splnění**.

Každý issue může nést informaci o své váze. **Váha** issue představuje jeho obtížnost, míru úsilí, které je třeba vynaložit pro jeho uzavření.

Některé issues má smysl **přiřazovat k milníkům** projektu. Je tak možné sledovat postup práce na řešení milníku a lépe předejít hrozícímu zpoždění.

Jelikož práce s issues ukládá veškerou **historii**, je možné sledovat to, jak dlouho v minulosti trvalo řešení podobných problémů, nebo vytíženost členů vývojového týmu.

Tím, že issues mají širokou škálu užití, je vhodné je od sebe odlišit. K tomu účelu se používají **štítky** (*labels*). Obvyklé typy issues jsou například:

- User Story,
- Feature,
- Engeneering,
- Bug,
- Question,
- Refactor,
- Documentation,
- Suggestion,
- Enhancement.

Štítky je navíc možné použít k popisu vlastností issues, např.:

- rozpracovanost (To Do, Doing, Done),
- priorita (High, Medium, Low),
- komponenta (Front End, Back End, ...),
- potřebné dovednosti (UX, DB, ...).

Z uvedených příkladů je zřejmé, že štítky jsou silným nástrojem pro týmový vývoj, který dostává smysl hlavně ve vazbě na zvolenou metodiku vývoje.

Podrobnosti můžete najít na stránce Issues a Lables v GitLab dokumentaci.

12.2 Workflow projektu

Workflow by mělo vycházet ze zvolené metodiky a představuje **postup provádění úkonů** v rámci **životního cyklu** softwaru. Ideálně tak zahrnuje práci s

- nápady,
- požadavky a *user stories*,
- vlastním vývojem,
- testováním a
- zpětnou vazbou z provozu aplikace.

Workflow se odráží v práci s issues, zejména za pomoci *kanban board* a štítků, které pomohou udržet informaci o tom, ve které fázi životního cyklu se issue právě nachází.

Velmi doporučené čtení o nastavení workflow můžete najít u Atlassian a podrobného průvodce workflow v blogovém příspěvku GitLab.

Příklad workflow z reálného projektu obsahuje tyto fáze:

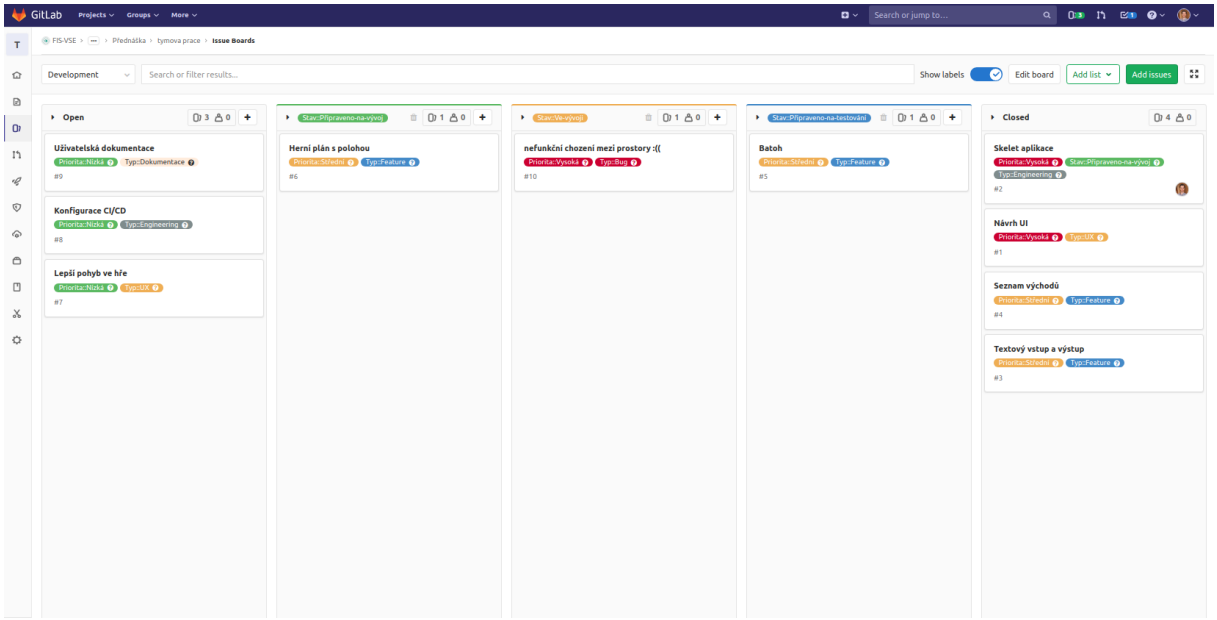
- nový návrh / problém,
- diskuze,
- připraveno pro vývoj,
- ve vývoji,
- revize kódu (peer review),
- schválení,
- slepé konce,
- odloženo (čeká se na určité podmínky),
- bug nelze zreprodukovat.

12.2.1 Kanban board

Kanban board (také *kanban tabule*) je nástroj, který se obvykle používá v řízení workflow agilního softwarového projektu. Je součástí filozofie štíhlé výroby, konkrétně přístupu Kanban.

Principem kanban tabule je vizualizace jednotlivých úkolů (*issues*), která napomáhá **zpřehlednit stav projektu** a zejména **snížit množství současně vykonávaných činností**. K tomu účelu je obvyklé stanovit limit počtu rozpracovaných činností. Limit rozpracovaných činností má smysl i vzhledem k náročnosti spojování více verzí zdrojového kódu do jednoho funkčního celku.

Využití tabule jednak zpřehlední informace o stavu projektu a jednak zjednoduší práci se štítky, protože štítky určitého issue se při jeho přesunutí do jiného sloupce změní samy.



Obrázek 12.2: Kanban board na GitLab.com

V praxi může mít projekt více než jednu tabuli. Pokud by tabule měla být jen jedna, bude nejvhodnější použít **fáze zvoleného workflow**. Další tabule se mohou zaměřit na:

- milníky projektu,
- přiřazené osoby,
- priority
- či další dle užitých štítků (dovednosti, komponenty aj.).

Kromě integrované tabule na GitLab.com je možné použít Trello nebo profesionální nástroj Jira.

O principu kanban tabule, výhodách fyzických tabulí a rozdílech mezi kanban a scrum tabulí si můžete přečíst v článku What is a kanban board? od Atlassian.

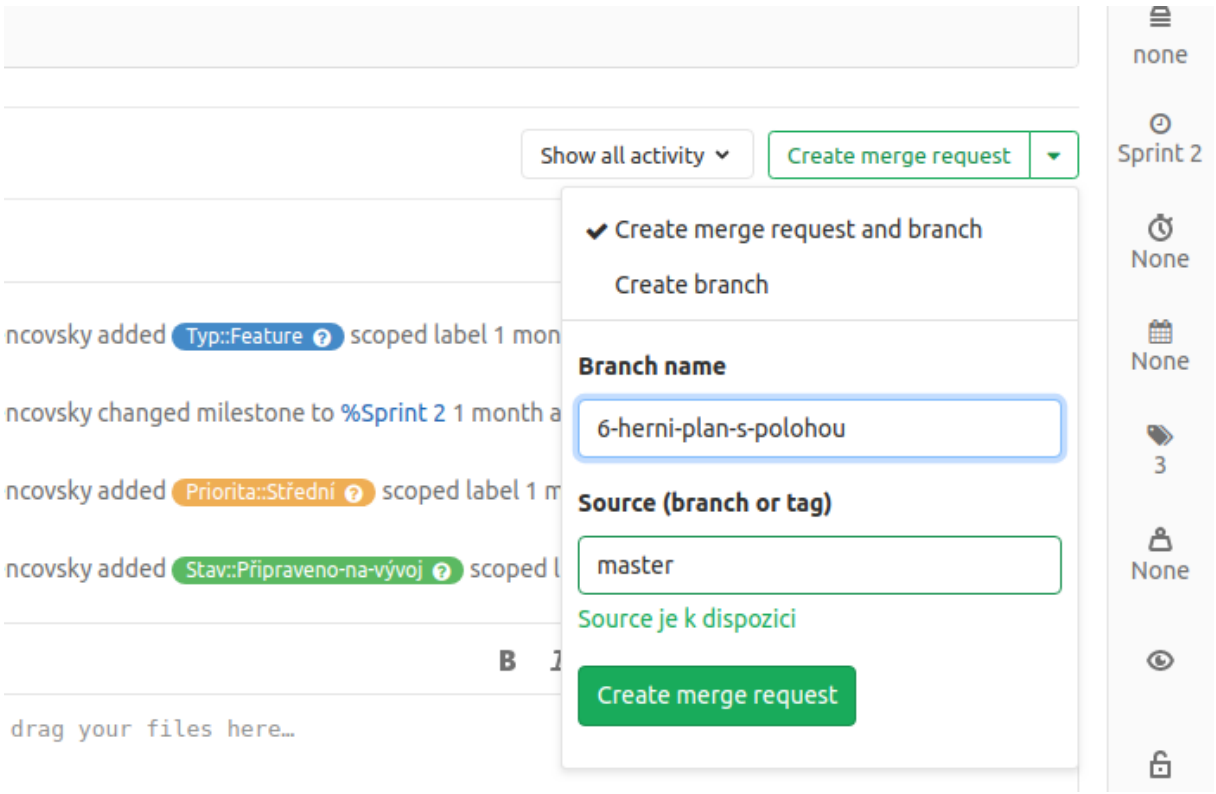
Podrobný návod pro použití kanban tabulí na GitLab.com najdete v dokumentaci v článku Issue Boards.

12.3 Práce s git větvemi

Práce s issues umožňuje i systematičtější práci s větvemi během vývoje. Každý issue totiž může být řešený ve vlastní větvi a po otestování se může spojit zpět s vývojovou větví.

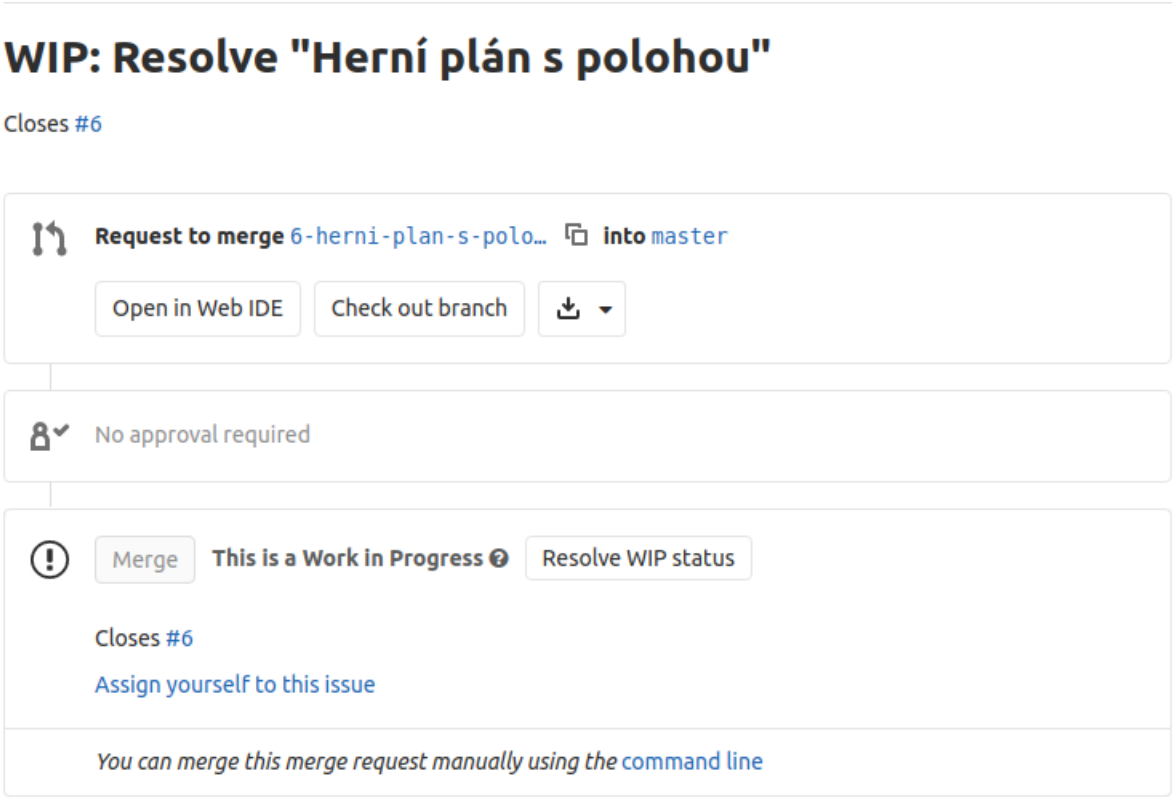
GitLab tuto praktiku podporuje tím, že umožňuje pro každý issue založit *Merge request*. Ten se zakládá kliknutím na zelené tlačítko *Create merge request*. GitLab automaticky předvyplní název nové větve podle čísla a názvu issue.

Doporučené je do názvu vložit také prefix podle typu issue, aby bylo jasné, jestli se jedná o opravu chyby (`fix`, `bugfix`) nebo vývoj nové funkcionality (`feature`, `story`). Na základě issue může být vytvořena například větev s názvem `feature/6-herni-plan-s-polohou` nebo `fix/10-nefunkcni-chozeni-mezi-prostory`. Jmenné konvence a zdrojová větev závisí na zvolené strategii pro *git workflow* (viz přehled strategií pro práci s větvemi od Atlassian).



Obrázek 12.3: Merge request na základě issue

Nově vytvořený merge request na GitLab nese označení *WIP*, které naznačuje, že práce stále probíhá a kód není připravený na testování a spojení s vývojovou větví.



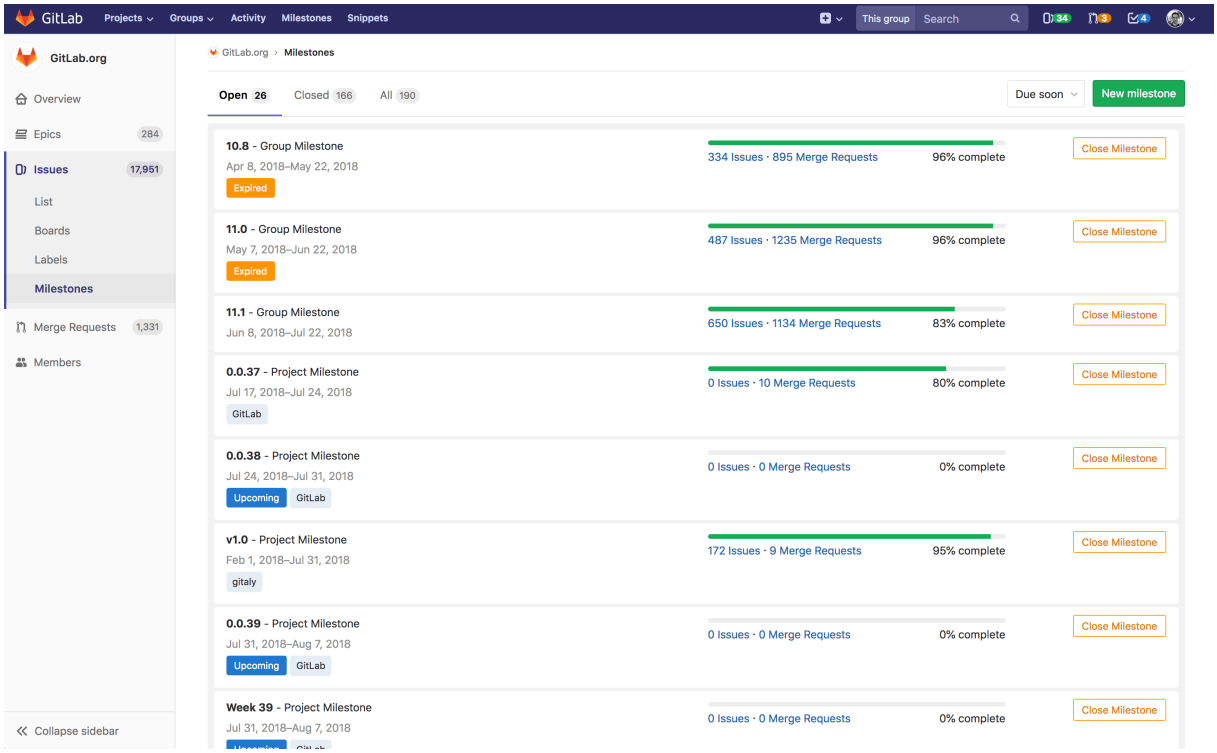
Obrázek 12.4: Status probíhající práce na issue

Stejně jako issue by měl každý request mít jednu osobu zodpovědnou za jeho splnění. Nadto je možné přidat podmínku, že spojení musí schválit konkrétní osoby nebo jen určitý počet osob. Tyto osoby jsou spoluodpovědné za otestování funkčnosti a stability změn v kódu.

Teprve ve chvíli, kdy je kód hotový, je možné kliknout na tlačítko *Resolve WIP status*. Tím se z názvu odebere prefix *WIP* a request bude možné spojit.

12.4 Milníky

Milníky (*milestones*) jsou pevné body v čase, které se stanovují v dostatečném předstihu. Definice milníku silně závisí na zvolené metodice. Milník může představovat naplánovaný sprint nebo verzi aplikace.

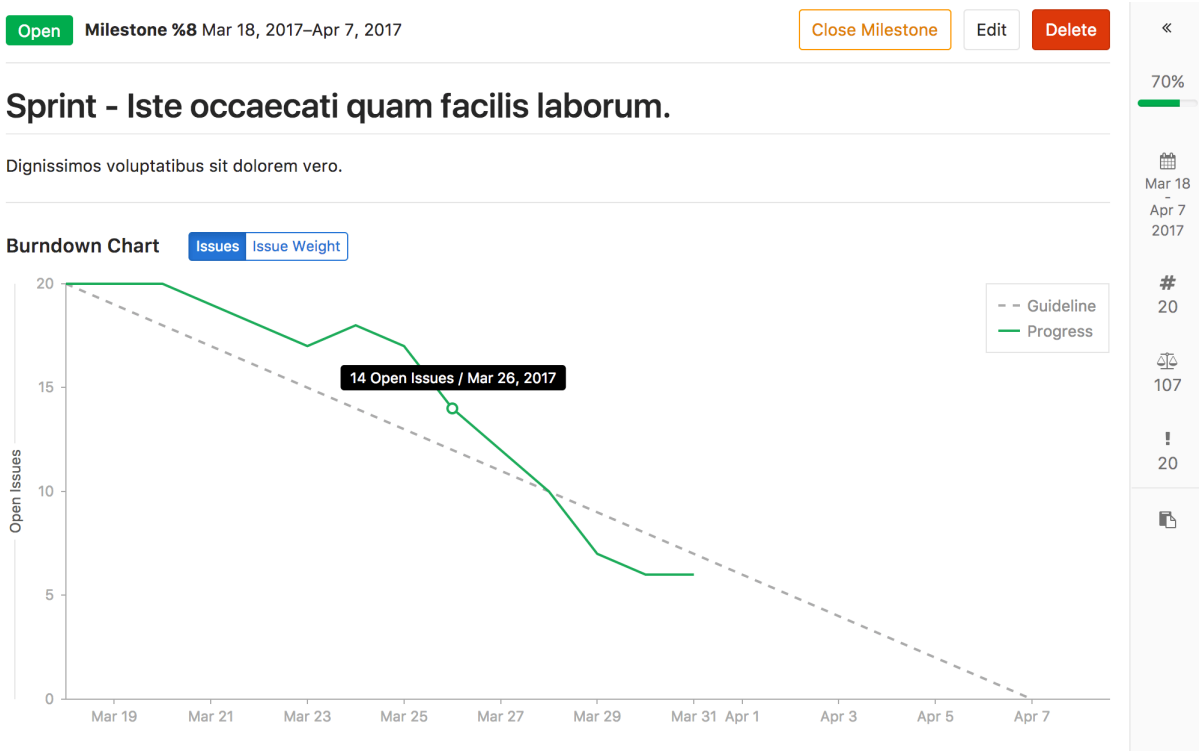


Obrázek 12.5: Přehled milníků projektu (zdroj: GitLab.com)

Milník může nést **informace o funkcionalitě nebo opravách**, které vlastník plánuje do aplikace zavést. Tyto informace se vkládají do popisu milníku.

S milníkem je **svázáno více konkrétních issues**, které je třeba před uplnynutím termínu dokončit.

Obvyklou technikou vizualizace vztahu issues k milníku je **burndown chart**. Vizualizace obsahuje časovou osu x a počet (nebo celkovou váhu) neuzavřených issues na ose y. Od data zahájení a počtu všech přiřazených issues vede orientační přímka k datu konce projektu, oproti které je možné porovnat aktuální stav.



Obrázek 12.6: Burdown chart (zdroj: GitLab.com)

Milníky je možné stanovovat a sledovat na úrovni projektu, ale i skupin projektů.

Více informací o práci s milníky se dozvíte na stránce Milestones v dokumentaci GitLab.

12.5 Týmová komunikace a integrace s nástroji na správu issues

Není třeba zdůrazňovat nutnost každodenní týmové komunikace. Na trhu existuje řada nástrojů, které komunikaci vývojářského týmu podporují. Práci na projektu může zefektivnit následující funkcionalita:

- Nezbytností je vyměňovat si části kódu. Nástroje proto umožňují **odesílat zdrojový kód** včetně zvýraznění syntaxe a možnosti ho jednoduše zkopírovat.
- Některé nástroje umožňují **integraci s nástroji na správu issues**. Přímo z konverzace je tak možné **zakládat nové issues** a volat další funkce. Propojení s aplikacemi *GitLab*, *Jira* nebo *Trello* nabízí například Slack. Více o propojení mezi aplikacemi GitLab a Slack se dozvíte z dokumentace na stránce Slack slash commands.

12.6 Příklady správy projektu z praxe

Příklady správy projektu z open source projektů na GitLab.com najdete v následujícím seznamu. U každého projektu věnujte pozornost použitým issues, štítkům, kanban tabulím, milníkům a strategii pro větvení.

- Chrome rozšíření pro Trello
- Blockchain platforma
- Generátor sylogismů
- Systém pro bezpečné přihlašování a připojení
- Galerie obrázků pro Android

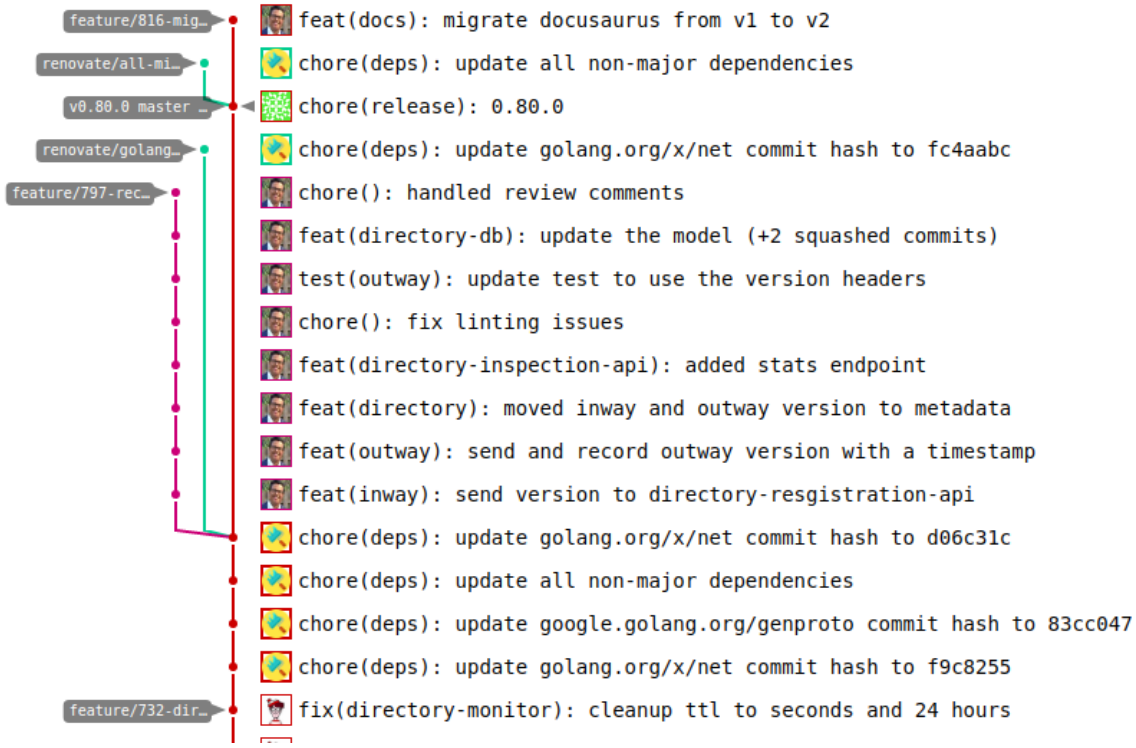
Kapitola 13

Release aplikace

Release (*vydávání* nebo *vydání*) je široký pojem označující aktivity vedoucí k transformaci **vyvíjené verze aplikace do verze připravené k použití**. Pojem může také označovat právě tu verzi aplikace, která transformací prochází nebo jí již prošla.

Pokud použijeme doporučený postup pro větvení kódu, vyvíjená verze aplikace se nachází v *develop* větvi. Ve chvíli, kdy vyvíjená verze aplikace obsahuje požadovanou funkcionalitu, je spojena s *release* větví. Práce na *release* větvi už nepřidávají žádnou novou hodnotu, ale zvyšují stabilitu aplikace. Verze aplikace v *release* větvi se někdy označuje jako RC (*Release candidate*). Více o strategii práce s větvemi se dozvíte v tutoriálu od Atlassian.

Po dostatečném otestování může být aplikace uvolněna pro použití a nasazena do provozu. Doporučené je spojit takto připravenou verzi s *master* větví pomocí *merge request* a označit příslušnou záložkou s čitelnou verzí hotové aplikace.



Obrázek 13.1: Graf verzí a release v master větvi (zdroj: NLX projekt na GitLab.com)

Další informace o vydávání aplikace si můžete přečíst v článcích Release a Stress-free release od Atlassian.

13.1 Pojmenování verze

Ačkoli má každá verze softwaru, která se vyvíjí v systému pro správu verzí, svůj hash, pro orientaci je nutné použít systém číslování, který bude srozumitelný pro vývojáře i uživatele.

V praxi se ukázalo, že nestačí jedno číslo verze, protože popis verze musí nést více informací. Minimálně to, jestli verze obsahuje **novou funkcionalitu** a jestli je dodržena **zpětná kompatibilita** API s předešlými verzemi.

Systém verzování, který obsahuje tyto informace, se nazývá sémantické verzování. Předepisuje, aby verze byla označena v tomto formátu:

`MAJOR.MINOR.PATCH-meta`

První číslo (*major*) označuje významnou změnu, která mění **zpětnou kompatibilitu rozhraní** (*API*) aplikace. V minulosti se ale často jednalo o marketingový tah, který měl vyvolat dojem vyšší vyzrálosti softwaru a potažmo měl za cíl vyrovnat se konkurenci, nebo

se jednalo o zevrubnější změny v aplikaci (viz Political and cultural significance of version numbers).

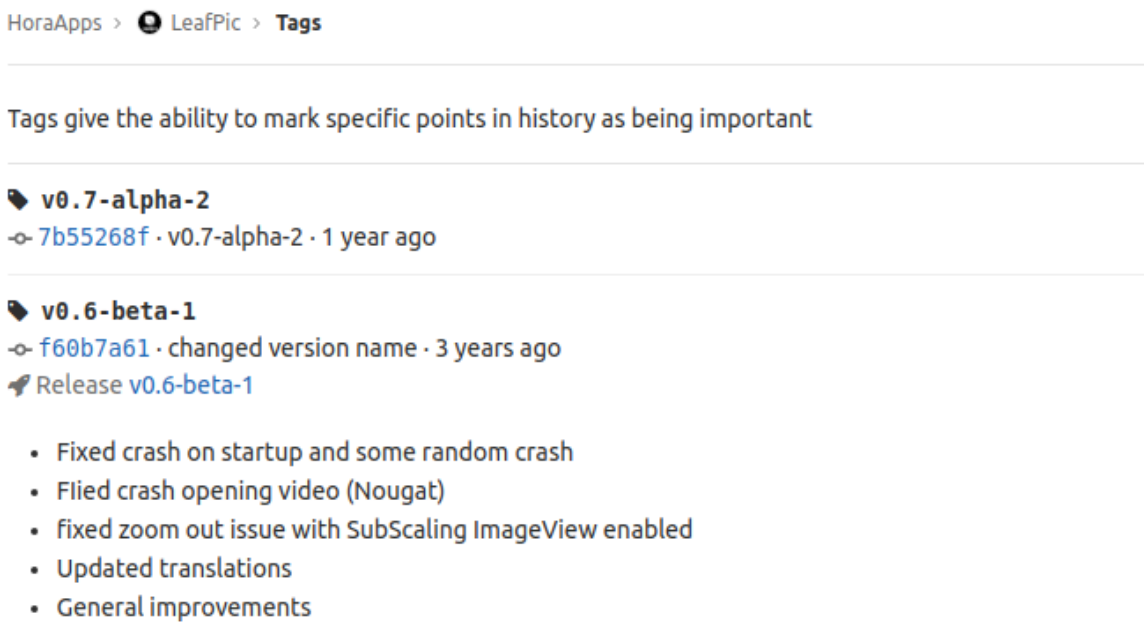
Druhé číslo (*minor*) se zvyšuje u všech dalších změn, které zpětnou kompatibilitu zachovávají a přinesly **novou funkcionalitu**.

Poslední číslo (*patch*) se zvyšuje při každé změně, která nemění zpětnou kompatibilitu a nepřináší ani novou funkcionalitu. Tyto změny pouze **zvyšují stabilitu aplikace**.

Název verze může být doplněn o **dodatečné informace (*meta*)** za pomlčkou. Nejčastěji se uvádí, že se ještě nejedná o hotový produkt, ale že se stále ještě vyvíjí (např. *SNAPSHOT*, *WIP*, *alpha*, *beta*).

13.2 Tvorba záložek na GitLabu

Seznam záložek (*tags*) se nachází na GitLab.com v menu *Repository* > *Tags*. Novou záložku je možné přidat kliknutím na zelené tlačítko *New tag*.



Obrázek 13.2: Seznam záložek (zdroj: LeafPic projekt na GitLab.com)

U nové záložky se vyplňuje **jméno** a **revize, pro kterou se záložka vytváří**. Pokud se jedná o release, je obvyklé použít jméno verze uvozené písmenem *v*, například *v1.0.0*. Označení verze by se mělo držet zmíněných zásad sémantického verzování.

Záložku je možné založit pro jakoukoli revizi v historii projektu, stačí **vyplnit hash revize**. Alternativně je možné napsat jméno větve a automaticky se bere poslední revize ve větvi.

K záložce je možné připojit **zprávu** (*message*) podobně jako ke každé revizi. Zpráva může krátce shrnout podstatu nové verze.

Další informace o verzi v podobě formátovaného textu je možné připojit poznámkou do *Release notes*. Do poznámky je vhodné shrnout všechny změny s krátkým popisem a připojit zkompilovaný otestovaný soubor s aplikací jako přílohu.

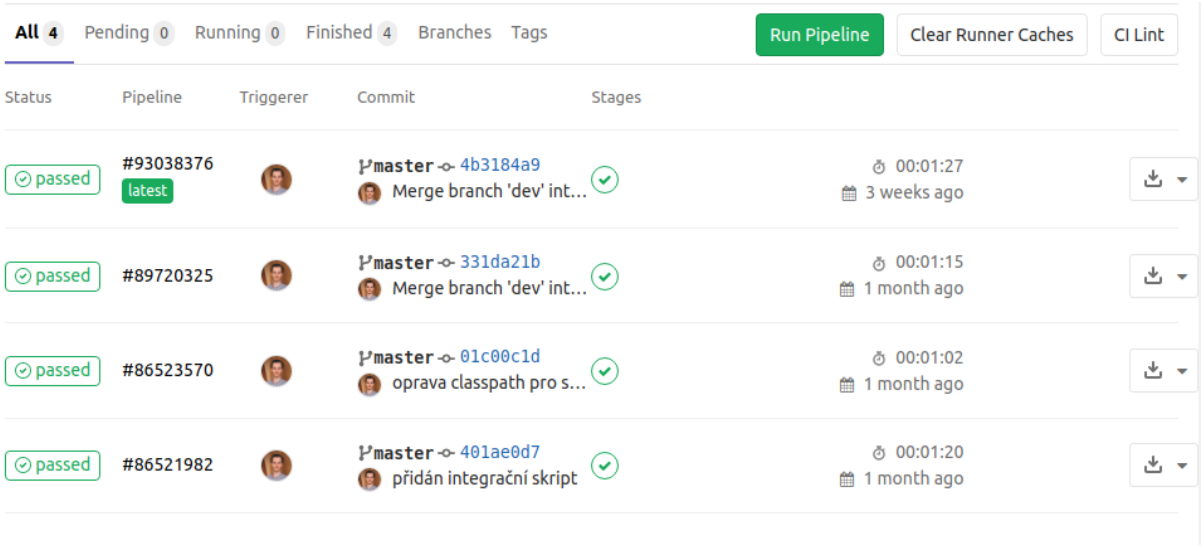
13.3 Kontinuální integrace a nasazení

S vydáváním softwaru úzce souvisí kontinuální integrace a nasazení (*Continual Integration / Continual Deployment, zkráceně CI/CD*). Pojem označuje **rutinní aktivity** vedoucí k testování změn aplikace, spojování do hlavních větví a sestavení doručitelného softwaru. Více informací najdete v článku Continuous integration vs delivery vs deployment.

Kontinuální integrace a nasazení je závislé na **automatizaci**. Po splnění určitých podmínek se spustí *pipeline*, která ve vybraném *kontejneru* vykoná **sled příkazů**. Jeden běh může být rozdělen do více fází (*job*) podle účelu, např. sestavení, testování a nasazení.

Podmínkou může být buď plánovaná **časová událost** (*schedule*), nebo ***commit*** do určité větve. Obvyklé je software sestavovat pravidelně nebo při nové revizi v *master* větvi.

Kontejner je balík softwaru, který je vložený do izolovaného prostředí, které obsahuje vše potřebné pro svůj běh. Například pro sestavení Java aplikace vyvíjené jako Maven projekt je zapotřebí kontejner, který obsahuje Java SDK a Maven. Více o kontejnerech se dozvíte v článku What is a Container?



Obrázek 13.3: Přehled sestavení na GitLabu

Shrnutí nejpoužívanějších řešení pro CI/CD najdete v článku Best 14 CI/CD Tools You Must Know.

Na **GitLabu** se integrace spouští automaticky podle konfigurace v souboru `gitlab-ci.yml`. Soubor se musí nacházet v kořeni repozitáře a mít validní *YAML* syntaxi, jinak nebude brán v úvahu. *YAML* je, podobně jako *XML* nebo *JSON*, syntaxe pro zápis dat v hierarchické struktuře. Soubor musí obsahovat odkaz na kontejner a sled příkazů organizovaných do jednotlivých fází.

Následující příklad konfigurace definuje jednu fázi, *sestavení*. Pod položkou `image` určí, že pro běh se použije kontejner obsahující Java SDK 8 a Maven s názvem `kaiwinter/docker-java8-maven`.

Do fáze `sestaveni` je vnořena položka `skript`, ve které je jediný příkaz, který zahájí testování jednotkovými testy a sestavení spustitelného archivu, `mvn install -B`.

Kromě toho obsahuje fáze `sestaveni` ještě položku `only`, která omezí běh *pipeline* jen na revize ve větvi *master*.

Poslední část se věnuje zachycení artefaktů. Do položky `artifacts` je třeba nakonfigurovat cestu k sestavenému archivu do položky `paths`. Po skončení běhu *pipeline* je možné archiv stáhnout, a to bez nutnosti kontaktovat vývojáře.

```
image: kaiwinter/docker-java8-maven

sestaveni:
 script:
 - "mvn install -B"
```

```
only:
- master

artifacts:
   paths:
   - target/adventura-cv-1.0.0.jar
```

Každý běh skriptu skončí buď úspěšně a je označen zeleným tlačítkem *passed*, nebo neúspěšně a je označen červeným *failed*. Po kliknutí na tlačítko se zobrazí podrobný výpis.

Více informací o kontinuální integraci a nasazení najdete v dokumentaci Gitlab.com.

Stránky našeho nakladatelství
https://oeconomica.vse.cz/

Název	**Nástroje pro tvorbu, správu a nasazování aplikací**
Autor	Ing. Filip Vencovský, Ph.D.
Vydavatel	Vysoká škola ekonomická v Praze Nakladatelství Oeconomica
Doporučeno	pro bakalářské studium na VŠE v Praze
Vydání	první
Návrh obálky	Daniel Hamerník, DiS.
Počet stran	104
DTP	Vysoká škola ekonomická v Praze Nakladatelství Oeconomica
Sazba	autor

Tato publikace neprošla redakční úpravou.

ISBN 978-80-245-2341-5

Zdarma ke stažení