Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Bakalářská práce

Vývoj aplikace pomocí low-code platformy

Jakub Vašák

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Jakub Vašák

Systémové inženýrství a informatika
Informatika

Název práce

Vývoj aplikace pomocí low-code platformy

Název anglicky

App development using low-code platform

Cíle práce

Hlavním cílem bakalářské práce je popsat metody a technologie, které se používají k vývoji aplikací na low- code platformě Outsystems. Dílčími cíli je vysvětlení pojmu ”low-code”, srovnání programování v low-codu s běžnými programovacími jazyky z pohledu časového a finančního hlediska, představení základních low- code platforem a dále návrh a implementace ukázkové aplikace v programu Outsystems. Ukázková aplikace byla vytvořena pro firmu SnT a zároveň bylo firmou dovoleno použít aplikaci pro bakalářskou práci.

Metodika

Bakalářská práce bude rozdělena na dvě základní části: teoretickou a praktickou. Teoretická bude založe- na na studiu literárních a internetových informačních zdrojů. Na základě zjištěných poznatků budou for- mulována teoretická východiska práce. Praktická část bude spočívat ve vytvoření aplikace na inventarizaci zařízení pro firmu SnT pro mobilní platformu Android. Aplikace bude umět číst čárové kódy jednotlivých instalovaných zařízení a ukládat je do databáze. Výsledná databáze se bude exportovat do souboru CSV.

Doporučený rozsah práce

35-40 stran

Klíčová slova

mobilní aplikace, low-code, Outsystems, Android, iOS, iPhone, software, objektové programování, progra- movací jazyk

---

Doporučené zdroje informací

Low-code Application Development – Mendix. Dostupné z: https://www.mendix.com/ Low Code Development Platforms A Complete Guide – 2020 Edition by Gerardus Blokdyk Lowcode platforma Outsystems. Dostupné z: https://www.outsystems.com/
The Forrester Wave – Low-Code Development Platforms. Dostupné z: https://www.appian.com/resources/the-forrester-wave-low-code-development-platforms-2017/?utm_source=email&utm_medium=drip&utm_campaign=low-code&utm_content=ar- forrester-lowcode-wave-2017

Předběžný termín obhajoby

2020/21 LS – PEF

Vedoucí práce

Ing. Jiří Brožek, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 23. 2. 2021

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 23. 2. 2021

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 15. 03. 2021

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Vývoj aplikace pomocí low-code platformy" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.03.2021 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Jiřímu Brožkovi, Ph.D. za pozornost a pomoc, kterou věnoval mé práci, a za jeho odborné rady, které pomohly k vypracování této práce. Dále bych rád poděkoval firmě S&T CZ s.r.o. za poskytnutí příležitosti pracovat s low-codem.

Vývoj aplikace pomocí low-code platformy

Abstrakt

Tato bakalářská práce se zabývá problematikou vývoje softwaru za pomocí low-code platformy a vytvoření ukázkové mobilní aplikace v platformě OutSystems. Obsah celé práce lze rozdělit do 3 hlavních částí.

První část bude zaměřena primárně na stručný přehled vývoje programování a následné seznámení s konceptem low-codu. Bude zde vysvětlen pojem low-code, zmíněna historie low-codu a postavení low-codu na trhu.

Ve druhé části bude vysvětlen pojem low-code platforma a následně představeny dvě vybrané platformy. Tato kapitola bude obsahovat i srovnání programování v low-code platformě s běžnými programovacími jazyky, a to jak z pohledu časového, tak i finančního.

Ve třetí části bude zkonstruována vzorová aplikace za použití platformy OutSystems. Tato vzorová aplikace bude vytvořena pro firmu S&T a bude sloužit pro čtení čárových kódu a následnou inventarizaci načtených zařízení. V této části budou také popsané uživatelské zkušenosti s používáním low-code platformy. Na závěr vyzdvihnu vybrané z nejlepší vlastnosti těchto platforem.

Klíčová slova: mobilní aplikace, low-code, low-code platforma, OutSystems, Mendix, Android, iOS, software, objektové programování, programovací jazyk

App development using low-code platform

Abstract

This bachelor thesis is to engage the issue of software development using a low-code platform and the creation of a sample mobile application in the OutSystems platform. The content of the thesis can be divided into three main parts.

The first part will focus primarily on a brief overview of the development of programming and subsequent acquaintance with the topic of low-code. The concept of low-code will be explained, the history of low-code and the position of low-code on the market will be mentioned.

The second part will explain the low-code platform and then introduce two selected low-code platforms. This chapter will also contain a comparison of programming in the low-code platform with common programming languages, both in terms of time and money.

In the third part, a sample application will be constructed using the OutSystems platform. This sample application will be created for the company S&T and will be used for reading barcodes and subsequent inventory of loaded devices. This section will also describe the user experience with using a low-code platform. Finally, I will highlight some of the best features of low-code platforms.

Keywords: mobile application, low-code, low-code platform, OutSystems, Mendix, Android, iOS, software, object-oriented programming, programming language

Obsah

1 Úvod

Již od úsvitu počítačové éry, kdy první počítače, které se objevily ve čtyřicátých letech 20. století, nabíraly obřích rozměrů, si vývoj počítačového softwaru do značné míry vyžadoval schopnost porozumět matematice, logice a jednomu nebo více programovacích jazyků, aby byl člověk schopen napsat funkční aplikaci. Obecně platí, že čím hlouběji člověk těmto pojmům rozumí, tím snazší bude pro něj tvořit kvalitní počítačový software, zatímco omezené porozumění by znemožnilo vykonat stejný úkol.

Dnes se však věci hodně změnily. Nástup grafických uživatelských rozhraní a výkonnějších strojů umožnil snahám o vývoj low-codu nabrat na obrátkách, což lidem usnadnilo navrhovat webové stránky, aplikace a dokonce multiplatformní systémy i s relativně malou znalostí počítačových věd.

Tato práce se bude zabývat tématikou low-code platforem a vysvětlením právě pojmu low-code. Jejím hlavním cílem je nastínění některých konceptů, obav a dostupných nástrojů v rámci vývoje low-code.

V této práci si také ukážeme několik low-platforem s největším zaměřením na platformu OutSystems. Právě na platformě OutSystems bude vytvořena mobilní aplikace pro inventarizaci zařízení pouze za pomocí grafického vývoje při psaní žádného nebo pouze minimálního kódu. Nakonec bude snaha o vyhodnocení, zda je možné ušetřit prostředky použitím takového způsobu vývoje namísto tradičních programovacích jazyků.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem bakalářské práce je popsat metody a technologie, které se používají k vývoji aplikací na low-code platformě OutSystems. Dílčími cíli je vysvětlení pojmu "low-code", srovnání programování v low-codu s běžnými programovacími jazyky z pohledu časového a finančního hlediska, představení základních low-code platforem a dále návrh a implementace ukázkové aplikace v programu OutSystems. Ukázková aplikace byla vytvořena pro firmu S&T a zároveň bylo firmou dovoleno použít aplikaci pro bakalářskou práci.

2.2 Metodika práce

Bakalářská práce je rozdělena na dvě základní části: teoretickou a praktickou. Teoretická část je založena na studiu literárních a internetových informačních zdrojů. Z důvodu rychlého vývoje tohoto oboru je ovšem většina informací čerpána z internetových zdrojů, a to převážně z dokumentací společností OutSystems a Mendix. Za pomoci uvedených zdrojů byla tato práce napsána a sestavena. Na základě zjištěných poznatků byla formulována teoretická východiska práce.

Praktická část se skládá z vytvoření aplikace na inventarizaci zařízení pro firmu S&T pro mobilní platformu Android. Aplikace umí číst čárové kódy jednotlivých instalovaných zařízení a ukládá je do databáze. Výslednou databázi je možnost exportovat do souboru CSV.

Na základě syntézy teoretických poznatků a výsledků praktické části pak byly formulovány závěry bakalářské práce.

3 Vývoj programování a programovacích jazyků

3.1 Programování a 1. generace programovacích jazyků

Předtím než se budeme zabývat přímo pojmem low-code a příslušnými low-code platformami, tak by bylo dobré si stručně popsat vývoj programovacích jazyků. Bude popsáno, jak jsme se dostali od prvních sekvenčních programovacích jazyků, jejichž typickým zástupcem byl například assembler, až po dnešní objektové programovací jazyky nebo právě zmíněné low-code vývojové platformy.

Co znamenají pojmy programovací jazyk a programování? Programovací jazyk je standardizovaný nástroj pro komunikaci s počítačem neboli zápis instrukcí ve zkrácené podobě srozumitelné člověku. Programování je zase proces tvorby programu, který programátor píše pomocí programovacího jazyka. Aby mohl být program vůbec spuštěn, tak se musí nejdříve načíst do operační paměti počítače (Stočes, 2019).

Pro první generaci je typické vytváření jednoúčelových programů, obvykle s využitím v matematice a fyzice. Za programovací jazyk se považoval strojový kód vyjadřující strojové instrukce, a se kterým pracují elektronické součástky. Programy a algoritmy byly zapisovány do paměti přímo v binárním kódu, který zpracovával příslušný procesor. Nutno podotknout, že počítače byly vybaveny prostředky pro převod binárního kódu na osmičkový nebo šestnáctkový kód a naopak, z důvodu lepší čitelnosti pro programátora (Kříž, 2002).

Takto vytvořené počítače a programy měly na svoji dobu relativně vysoký výkon, ale sloužili hlavně jako automaty, respektive měli pevně dané instrukce, které uměly vykonávat. Jednotlivé instrukce kódu měly přesně dané místo v operační paměti, ve které byly uloženy. Dalším problémem bylo hledání chyb a úpravy programů. Navíc tématika strojového jazyka byla hardwarově závislá na jednotlivých komponentech, což znamená, že jeden a tentýž program měl na různých počítačích vždy různý kód. V dnešní době se tento způsob programování objevuje jen ojediněle. (Kučera, 2000)

Nutno podotknout, že první počítače neobsahovaly polovodičové součástky, a to zejména tranzistory. Ty následně daly základ pro rozvoj programovacích jazyků.

3.2 Jazyk symbolických adres – 2. generace

S rozvojem výpočetní techniky bylo zřejmé, že programování ve strojovém kódu je značně neefektivní a zdlouhavé. Prvním krokem od strojového kódu bylo vytvoření jazyka symbolických adres, který místo číselných informací využíval symbolických názvů instrukcí a současně nepožadoval absolutní alokaci paměti, protože se místo nich využívalo adres symbolických. Symbolické instrukce se následně musí přeložit zpět do strojového kódu. Překladač jazyka symbolických instrukcí se nazývá Assembler. Assembler také umožňoval přiřazovat symbolickým adresám adresu fyzickou. Název Assembler se často chybně označuje za jazyk symbolických adres (Kučera, 2000; Tišnovský, 2019).

Tento typ programovacího jazyka se také uvádí jako nízkoúrovňový. Nízko úrovňový jazyk je typ programovacího jazyka, který obsahuje základní pokyny rozpoznané počítačem, kdy označení nižší právě odkazuje na velmi malý rozdíl mezi daným programovacím jazykem a strojovými instrukcemi procesoru. (Tišnovský, 2019)

Další velkou výhodou využití jazyka symbolických adres bylo zavedení tzv. makra, čímž programátor může zaměnit dlouhou sekvenci příkazů jediným krátkým výrazem. Pozdější verze Assembleru využívaly nástroj debuggeru, který umožňuje zastavení programu na daném místě nebo krokovat proměnné (Kříž, 2002).

Největší nevýhodou jazyka symbolický adres je jeho přetrvávající strojová závislost a stále velká obtížnost programovaní, která spočívá ve vysokém počtu jednotlivých příkazů a malé přehlednosti. Programátor do té doby musel být také odborníkem na hardware. (Tišnovský, 2019)

V současné době má tento nízkoúrovňový programovací jazyk stále využití pro přímé ovládání hardwaru, konkrétně při tvorbě částí operačních systémů nebo ovladačů, kde je kladen vysoký důraz na rychlost, efektivitu a malou paměťovou náročnost.

3.3 Vyšší programovací jazyky – 3. generace

Následný vývoj programovacích jazyků se hlavně zaměřil na odklon od nižších programovacích jazyků. Jedním z hlavních cílů bylo přiblížit programování více k algoritmickému myšlení člověka, aby mohl programátor psát takové algoritmy, které by odrážely povahu řešených problémů. Do této doby programy pouze odrážely technickou

realizaci výpočetních procesů v počítači. Jako další cíl bylo odstranění strojové závislosti jazyků, což by v praxi značilo nezávislost na konkrétním hardware a snadnou přenositelnost napsaných programů. (Kučera, 2000)

Jako jeden z prvních takových programovacích jazyků byl jazyk FORTRAN (z anglického FORmula TRANslator), který vnikl v roce 1957 ve společnosti IBM pod vedením Johna Backuse. Jedna z největších výhod FORTRANu byla možnost rychlého naučení jazyka a efektivnost spustitelného kódu. Tento jazyk byl především orientován na vědecko-technické výpočty a je v jeho inovované verzi využíván do dnes. (Hájek, 2007)

Následoval programovací jazyk ALGOL (58 následně 60 a 68), který jako první poskytoval snadno pochopitelnou a jednotnou množinu jazykových prostředků pro popis algoritmů. Ačkoliv samotný jazyk nebyl příliš populární, dal inspiraci za vznik dalším programovacích jazyků jako jsou Pascal nebo C (Tišnovský, 2020).

Základ jazyka COBOL (z anglického COmmon Business Oriented Language) byl vytvořen v roce 1959, když se na konferenci pořádané v USA sešli zástupci výrobců počítačů s ministerstvem obrany USA za účelem na vytvoření standardu vysokoúrovňového programovacího jazyka. Výsledkem práce této komise se stal jazyk COBOL, u kterého došlo v 1960 k masovému rozšíření z důvodu instalace COBOLu na každém komerčně instalovaném PC. Jednou z jeho vlastností byla, krom vědecko-technických výpočtů, i relativně dobrá čitelnost pro uživatele – neprogramátory, protože zápis programu se podobal běžnému anglickému textu (Tišnovský, 2009).

Dále by bylo dobré zmínit jazyk BASIC (Beginner's All-purpose Symbolic Instruction Code), který kladl důraz na co největší jednoduchost a využíval se na výuku programování. Později tento jazyk našel uplatnění hlavně v mikropočítačích. (Kučera, 2000)

3.3.1 Strukturované programování – pokračování 3. generace

Nyní je důležité rozlišit strukturované a sekvenční programování. U sekvenčního programování se jednotlivé příkazy vykonávají tak, jak po sobě následují ve zdrojovém kódu, popřípadě je zde příkaz skoku, který přesune vykonávání programu na jiný řádek. Jazyk symbolických adres i zdrojový kód jsou zástupci sekvenčního programování. (Čada, 2009)

U strukturovaného programování se algoritmus rozděluje na dílčí úlohy, které se poté zpracovávají postupně a až v závěru je spojíme v jeden celek metodou shora dolů. Dílčí úlohy by měli požívat základní 3 řídící struktury – sekvence, selekce a opakování. Pokud možno, tak se strukturované programování vyhýbá příkazu skoku goto. Za tvůrce tohoto konceptu se považuje Edsger W. Dijkstra se svým článkem „Goto statement considered harmful“ (Polanský, 2012; Čada, 2009).

Z myšlenek strukturovaného programování vzešel na konci 60. let nový programovací jazyk – Pascal. Tvůrcem tohoto velice populárního strukturovaného jazyka je Dr. Niklaus Wirth ze Švýcarska. Pascal byl založen na jasných, srozumitelných a strukturovaných příkazech, díky čemuž se dodnes využívá na školách jako ideální jazyk pro výuku programování. (Polanský, 2012)

Ovšem nejznámějším jazykem z toho období se stal programovací jazyk C, který je úzce spjat s operačním systémem UNIX. Vznikl v 70. letech v Bellových laboratořích AT&T a jeho tvůrci jsou Dennis Ritchie a Brian Kerningham. Jazyk C je strukturovaný, úsporný a velmi výkonný. Díky své obrovské univerzálnosti je vhodný jak na psaní systémových programů (dokonce je v něm naspaný operační systém UNIX), tak na profesionální programy. C je typickým zástupce kompilačních jazyků, jehož zdrojový kód je vstupem do kompilátoru (překladače). Kompilátor provede syntaktickou kontrolu celého kódu a teprve potom je přeložen do strojového kódu a spuštěn jako samostatný soubor. Jazyk C dal základ pro vznik jazyků C++ nebo C# (Černohorský, 2003; Čada, 2009).

3.3.2 Objektové programování – pokračování 3. generace

Objektové programování navazuje na strukturované programování, kdy se snaží překonávat hierarchii strukturovaného programování a modelovat řešení úloh principy reálného světa. Jednou ze základní snah objektově orientovaného programování byla koncepce znovupoužitelnosti. Takový kód by zajistil snadnou přenositelnost mezi projekty a opravitelnost, kde by stačilo opravit chybu pouze na jednom místě. Základní jednotkou objektového programování je objekt, který představuje nějaký objekt z reálného světa. Objekt má své atributy, neboli data, které uchovává a odlišuje se tím od ostatním objektů. Dále má také metody, čímž jsou myšleny vlastnosti objektu, které umí program vykonávat. Jednotlivé objekty spolu dynamicky komunikují na základě událostí. (Čada, 2009)

Objektově orientované programování se poprvé objevilo na přelomu 60. a 70. let 20. století. Prvním programovacím jazykem využívajícím tento přístup se stal Simula 67 (SIMple Universal programming LAnguage). Tento jazyk dal následně za vznik programovacímu jazyku Smalltalk, který jako první zavedl úplnou implementaci pojmu objektově orientované programování. Smalltalk byl vyvinut na počátku 70. let ve firmě Xerox. Popularizací jazyka Smalltalk se začínalo objektové programování dostávat do povědomí lidí a začínali vznikat tzv. objektové nástavby. Objektová nástavba prakticky znamená sloučení dvou zavedených jazyků. Jako typického zástupce bych mohl uvést jazyk C++, který vznikl spojením jazyků Simula 67 a C (Pecinovský, 2009; Čada, 2009).

Autorem C++ je Bjarne Stroustrup, který začal na jazyku pracovat začátkem 80. let s původním označením „C with Classes“. Největší výhodou C++ je spojení rychlosti jazyka C s možnostmi objektového programování. Díky této vlastnosti je možné kompilovat programy a knihovny, které byly původně navrženy pro platformu C a zároveň využívat objektovou nadstavbu, díky které můžeme části kódu rozdělit na komunikující objekty. V současné době je C++ jedním z nejpoužívanějších programovacích jazyků. (Pecinovský, 2009)

3.4 4. generace programovacích jazyků

Čtvrtá generace programovacích jazyků se snažila zase o něco přiblížit lidskému jazyku. Charakteristickým znakem této generace je pokus o zrychlení práce programátora a celkově zjednodušení komunikace s počítačem. Častěji se využívaly příkazy podobné mluvenému anglickému slovu nebo používání základního grafického prostředí. Programy bývají napojeny na databáze, se kterými dále pracují, a tak umožňují efektivnější vývoj podnikově orientovaných systémů (Stočes, 2019).

Existuje celá řada jazyků 4. generace, které se svojí funkcionalitou zaměřují na různou problematiku. Některé obsahují možnost generovat nějaký kód jazyka 3. generace nebo naopak takový kód zpracovávat, dále jazyky zaměřené na generování zpráv, matematickou optimalizaci, vývoj grafického uživatelského rozhraní nebo vývoj webových aplikací. Veškeré low-code platformy spadají pod 4. generaci programovacích jazyků.

Jako příklad jazyka 4. generace byl vybrán jazyk Sequel (Structured English Query Language), který vznikl roku 1974 v laboratořích společnosti IBM. V 80. letech bylo jméno

zkráceno pouze na SQL (Structured Query Language). Jedná se strukturovaný dotazovací jazyk, který měl být co nejbližší přirozenému jazyku, přičemž slovem dotazovací se myslí, že se jedná o jazyk specializovaný pro manipulaci daty. V SQL tedy nelze naprogramovat aplikaci využívající uživatelské rozhraní či komunikaci s jinými programy, proto je třeba kombinovat SQL s jinými programovacími jazyky (Vostrovský, 2014).

Cílem tohoto jazyka je vývojářům poskytnout standardní metodu přístupu k datům uloženým v databázovém systému, která by byla nezávislá na dalších použitých vývojových nástrojích. SQL příkazy lze použít pro přímou práci s databází nebo mohou být použity jiným programovacím jazykem a vytvořit tak databázové rozhraní. Všechny hlavní systémy pro správu databází tento jazyk podporují. (Kocan, 1998)

Jazyk SQL můžeme rozdělit na dvě části. Za první část se považuje DDL (Data Definition Language), díky kterým můžeme nadefinovat vlastní databázové struktury. Pomocí druhé části zvané DML (Data Manipulation Language), můžeme s daty manipulovat, respektive data vkládat, upravovat je nebo vytvářet dotazy (Vostrovský, 2014).

Nyní by bylo dobré si rozdělit programovací jazyky na procedurální (imperativní) a neprocedurální (deklarativní). U procedurálních programovacích jazyků je program popsán nějakou posloupností příkazů, kdy příkazem se myslí provedení určité operace. Pomocí procedurálních jazyků prakticky říkáme, jak chceme konkrétní věc provést. Data jsou zde ukládána pomocí proměnných, které se ukládají do paměťových míst. Většina programovacích jazyků využívaných v dnešní době jsou procedurálního typu. (Čada, 2009)

Neprocedurální programovací jazyky jsou oproti tomu jazyky, kde sice specifikujeme řešený problém, ale nepopisujeme přesný způsob řešení daného problému. Pomocí procedurálních jazyků prakticky říkáme, co chceme provést. Data nebo hodnoty se zde nepředávají pomocí proměnných, ale ve formě návratové hodnoty nějaké funkce. Jako příklad deklarativního jazyka můžeme uvést právě zmíněné SQL. (Žoltá, 2013)

3.5 Přirozené jazyky – 5. generace

Tato generace je reprezentována neprocedurálními programovacími jazyky, především funkcionálním a logickým programováním. V předchozích generacích musel programátor vytvářet algoritmus, který popisoval, jak daný problém vyřešit. Naproti tomu jazyky 5. generace se zabývají tím, co se počítá. Jak uvádí Žoltá „Program je chápan

jako funkce, která na základě vstupů vrátí výstup. Stejná funkce se stejnými argumenty dává u deklarativního programování vždy stejný výsledek (vyhýbají se používání globálních proměnných)“ (Žoltá, 2013).

Tímto způsobem programátor v podstatě pouze definuje objekty, pravidla, omezení a podmínky, které musí řešení splnit. Počítač následně vybere nejvhodnější algoritmus pro dosažení požadovaného řešení.

Jedním z klíčových pojmů neprocedurálního programování je technika zvaná rekurze. Rekurze se dá jednoduše popsat jako funkce, která volá sama sebe, a to v době, kdy předchozí volání nebylo ještě dokončeno. Po každém volání rekurze by mělo dojít ke zjednodušení problému. Největší výhodou správně implementované rekurze je tvorba krátkého a efektivního algoritmu. Naopak nevýhodou je, že lze aplikovat pouze na některé typy úloh a její vyšší paměťová náročnost. (Kříž, 2002)

3.5.1 Funkcionální programování

Základ funkcionálního programování pochází z teorie funkcí tzv. lambda kalkulu, kde jsou všechny konstrukce funkcemi. Program je zde zapsán ve formě výrazu. Jak uvádí Škvarda, „Nejdůležitějšími složkami těchto výrazů jsou funkce a jejich aplikace na argumenty. Výpočet funkcionálního programu spočívá ve zjednodušování výrazu až do doby, kdy výraz dále zjednodušit nelze. Tento dále nezjednodušitelný tvar je výsledkem výpočtu“ (Škvarda, 2006).

Největší výhodou funkcionálních jazyků je vyšší míra abstrakce než u klasických procedurálních jazyků. Vyšší úroveň abstrakce znamená stručnější vyjádření a snazší rozdělení do komponent. To způsobí i rychlejší detekci chyb a snazší ladění. Za nevýhodu se považuje vyšší výpočetní náročnost a občasné horší čtení programu (Pekař, 2019).

Funkcionální programovací jazyky můžeme rozdělit do dvou skupin. První skupinou jsou čistě funkcionální jazyky. To jsou takové jazyky, které pouze dodržují principy funkcionálního programování. Jako druhou skupinu jsou hybridní jazyky, které podporují krom funkcionální programování i procedurálním programováním. (Žoltá, 2013; Almásy, 2001)

Za první funkcionální jazyk se považuje LISP (LISt Processing), který by vytvořen na MIT Johnem McCarthym v roce 1960. LISP se považuje za hybridní jazyk. První verze

jazyka byla velice jednoduchá, obsahovala pouze několik operaci, ale i přes to byla Turingovsky kompletní. Tento programovací jazyk byl od začátku využíván v oblasti implementace umělé inteligence, což není náhodou, protože samotný John McCarthy je tvůrcem termínu umělá inteligence. Jazyk LISP se v různých dialektech využívá do dnes (Tišnovský, 2019).

Příkladem moderního čistě funkcionálního jazyka je jazyk Haskell, který vznikl v roce 1990 a vychází z jazyka Miranda. Haskell se považuje za líný jazyk, což znamená, že jazyk počítá pouze to, co je potřeba k získání výsledku. (Almásy, 2001)

3.5.2 Logické programování

Logické programování je založeno na výrokové logice. Program je zde vyjádřen relacemi, které jsou popsány pomocí logických predikátů. Příkladem logického programování je programovací jazyk Prolog. (Žoltá, 2013)

Prolog (PROgraming in LOGic) vznikl v roce 1972 ve Francii. Tento programovací jazyk je založen na predikátové logice I. řádu a Hornových klauzulích. Programy zde fungují na základě zadávaných faktů (predikátů) a pravidel, které následně vytvoří tzv. Prologovskou databázi. Takže v Prologu definujeme nějaké tvrzení, na které se pak dotazujeme. Pokud existuje řešení vyhovující námi zadanému systému klauzulí, Prolog je nalezne. (Kříž, 2002; Žoltá, 2013)

Tento způsob programování je využíván především pro vědecké účely.

3.6 Vizuální vývoj aplikací – Low-code

Autorkou následující velice výstižné definice low-codu je Alexander Forsyth. „Low-code je přístup k vývoji softwaru, který umožňuje doručování aplikací rychleji a s minimálním ručním kódováním. Low-code platformy jsou souborem nástrojů, které umožňují vizuální vývoj aplikací prostřednictvím modelování a grafického rozhraní. Low-code umožňuje vývojářům přeskočit ruční kódování a zrychlit proces přechodu aplikace do výroby“ (Forsyth, 2021).

Low-code je výraz pro vývoj webu, aplikací nebo softwaru, který nevyžaduje velkou znalost programování. Takový vývoj je obvykle uskutečněn pomocí šablon a uživatelsky přívětivého a intuitivního rozhraní typu drag-and-drop, který umožňuje vývojářům zrychlit

a zlehčit tvorbu požadované aplikace. Low-code nabízí skrze své platformy nástroje, které umožňují zmíněný vizuální vývoj aplikací. O low-code platformách bude více řečeno později. Jelikož samotný low-code je jednodušší než tradiční programování, je také snazší psát přehlednější kód a ladit jakékoli problémy. (Forsyth, 2021)

Uživatelé, kteří mají pouze malé zkušenosti s programováním, mohou využít vývojové nástroje poskytnuté low-codem, k vytvoření relativně složitých aplikací s velkým množstvím funkcí. Navíc vývojáři mající zkušenost s jedním nebo více programovacích jazyků, se mohou snadno přizpůsobit modelové logice.

3.6.1 Historie

Platformy pro vývoj low-code aplikací usnadnily a zefektivnily programování. Namísto psaní stovek tisíc řádků kódu pro vytvoření aplikace tyto platformy umožnily vizuální pracovní postupy.

Jak bylo zmíněno, low-code patří do 4. generace programovacích jazyků. V roce 1982 James Martin publikoval knihu zvanou Application Development Without Programmes. V této knize se objevila myšlenka vizuálního programování. Takový nápad se samozřejmě uchytil a nedlouho potom vznikl pojem rapid application development (RAD). Bohužel dřívější řešení nebyla optimální, nabízela pouze rozhraní, která jednoduše zakrývala skutečný kód generující aplikaci. Nástroje, které tyto rozhraní nabízely, byly zaměřené na rychlý, interaktivní vývoj s omezenou funkčností, tudíž vytváření složitějších aplikací bylo velice náročné. (Wilfrid, 2020)

Počátky vývoje low-codu, tak jak ho známe dnes, lze vysledovat až do roku 2011, kdy byla vydána zpráva o nových platformách pro vývoj vlastních aplikací. Nicméně až v roce 2014 byl vytvořen termín „Low-code“. Tento termín byl vytvořen společností pro průzkum trhu, zvanou Forrester, při vydání zprávy „New Development Platforms Emerge For Customer-Facing Applications“, která popsala tento nový fenomén a vytvořila termín „low-code“. Low-code byl popsán jako termín pro klasifikaci vývojových platforem, které se zaměřují na jednoduchost a snadné použití. Tyto platformy umožňovaly vývojářům a uživatelům všech úrovní dovedností programovat aplikace, aniž by museli přímo znát programování. (Richardson, a další, 2014; Dearmer, 2021)

Kolem této doby společnost Gartner pro výzkum a poradenství v oblasti IT vytvořila termín „citizen-developer“, definovaný jako „uživatel, který vytváří nové obchodní aplikace pro spotřebu ostatních, pomocí vývojových prostředí schválených podnikovým IT“ – jinými slovy někdo, kdo vytváří obchodní aplikace pro interní spotřebu, aniž by byl součástí vývojářského týmu (Dearmer, 2021).

Low-code platformy nabízejí, na rozdíl od svých předchůdců, samostatné prostředí, které umožňuje vytvořit komplexní aplikace. Dalším posunem je cloudové řešení platforem. Díky tomu uživatelé se nemusí bát aktualizací. Další výhodou je vyšší zabezpečení. Pokud samotná platforma nabízí ochranu, která je na vysoké úrovni, následný čas k samotnému bezpečnému nasazení aplikace do provozu je mnohem kratší (Wilfrid, 2020).

3.6.2 Trh

Low-code trh se čím dál tím více zvětšuje. Společnost Gartner předpovídá, že „Do roku 2023 přes 50 % středních a velkých podniků přijme aplikační platformu Low-code jako jednu ze svých hlavních strategických aplikačních platforem“ (Vincent, a další, 2020).

Jak se uvádí na stránkách společnosti OutSystems „Do roku 2024, low-code bude zodpovědný za více jak 65 % vyvíjených aplikací a reakce na pandemii COVID-19 tento tlak na poskytování digitálních řešení pouze zvýší.“ (Forsyth, 2021)

Jak bylo zmíněno, trh s low-code platformami nadále roste s velkým zájmem klientů jak pro vývoj aplikací, tak pro další použití související s aplikacemi, jako je integrace a cloud služby. Za leadery na low-code trhu se dají považovat společnosti Salesforce, Microsoft, OutSystems a Mendix. (Vincent, a další, 2020)

Velké množství platforem nabízí free verze, kde si můžeme zkusit vývoj zkušební aplikace v low-code prostředí. Většina platforem požaduje měsíční nebo roční poplatky za využívání jejich programovacího prostředí. Vývojová prostředí low-code platforem se liší podle jednotlivých vydavatelů.

Jako příklad byly vybrány dvě low-code platformy. Jedna z těchto platforem byla využita při vytváření ukázkové aplikace. Důvod výběru těchto platforem byly následující:

- Obě platformy se nachází v kvadrantu Leaders a považují se za jedny z nejlepších na trhu (obrázek 3.1)
- Obě platformy nabízí zdarma verze, které může vyzkoušet každý
- Poslední důvodem byla osobní zkušenost s platformami díky společnosti S&T

Obrázek 3.1: Gartner – Low-code Application Platforms (Vincent, a další, 2020)

Popis jednotlivých kvadrantů z obrázku 3.1:

- Leaders – firmy, které mají jak dobré provedení (technicky dobrý produkt odpovídající dobrým prodejům), tak i vizi (inovace produktu, která odpovídá marketingové strategii)
- Challengers – firmy, které mají dobré provedení, ale horší vizi

- Visionaries – firmy, které mají dobrou vizi, ale horší provedení
- Niche Players – firmy, které nemají dostatečně dobrou vizi a provedení, aby se dostaly do ostatních kvadrantů. Tyto prodejci mohou být dobrou volbou pouze pro konkrétní scénáře, hlavně díky jejich lepšímu cenovému modelu (Vincent, a další, 2020)

4 Low-code platformy

Jak již bylo uvedeno, low-code platformy umožňují vytvořit jednoduché aplikace bez potřeby většího porozumění v oblasti programování a programovacích jazyků. Uživatelé mají možnost vytvářet své aplikace jednoduchým skládáním nebo úpravou předdefinovaných částí (bloků) v grafickém prostředí jednotlivých platforem. To umožňuje uživatelům vyvíjet zcela funkční aplikace pro své nebo firemní potřeby. Na tyto platformy může být také nahlíženo jako na něco co je mezi vlastním vytvářením specializovaného softwaru pomocí tradičního vývoje a zakoupením již hotové aplikace. Ovšem Low-code platformy jsou více založeny na vlastním vytváření aplikací.

Samotné platformy jsou především přizpůsobeny pro zjednodušení tvorby firemních aplikací. To znamená, že je zde primární zaměření na vývoj a design uživatelských prostředí webových nebo mobilních aplikací, firemních operací nebo příslušných databází. Samotné programování v low-code platformách je sice sníženo na minimum, avšak v nějaké míře se zde pořád nachází. Kód může být vynucen při větších úpravách funkcionalit nebo pro vyřešení neobvyklých situací.

Jak uvedl Malcom Rose „Low-code platformy obsahují out-of-the-box funkce, které může vývojář může snadno a bezproblémově využít. Představte si tyto předem připravené komponenty jako krabici Lega. Každý blok má konkrétní účel a může být použit v různých aplikacích jako kousek větší skládačky. Například vývojáři by nemuseli kódovat objekty uživatelského rozhraní od nuly, když si mohou jednoduše vybrat z předem připravených komponent uživatelského rozhraní. Navíc tyto předem připravené komponenty lze snadno překonfigurovat a aktualizovat dle potřeby.

Uživatelé hodnotící low-code platformy by se měli pečlivě podívat na knihovnu předem připravených komponent a funkcí, které platforma poskytuje. Pokud je například cílem vybudovat systém pro správu podnikových smluv, budete chtít hledat funkce jako je správa dokumentů, správa obchodních procesů, obchodní pravidla a schopnost dynamicky generovat smluvní dokumenty PDF“ (Ross, 2018).

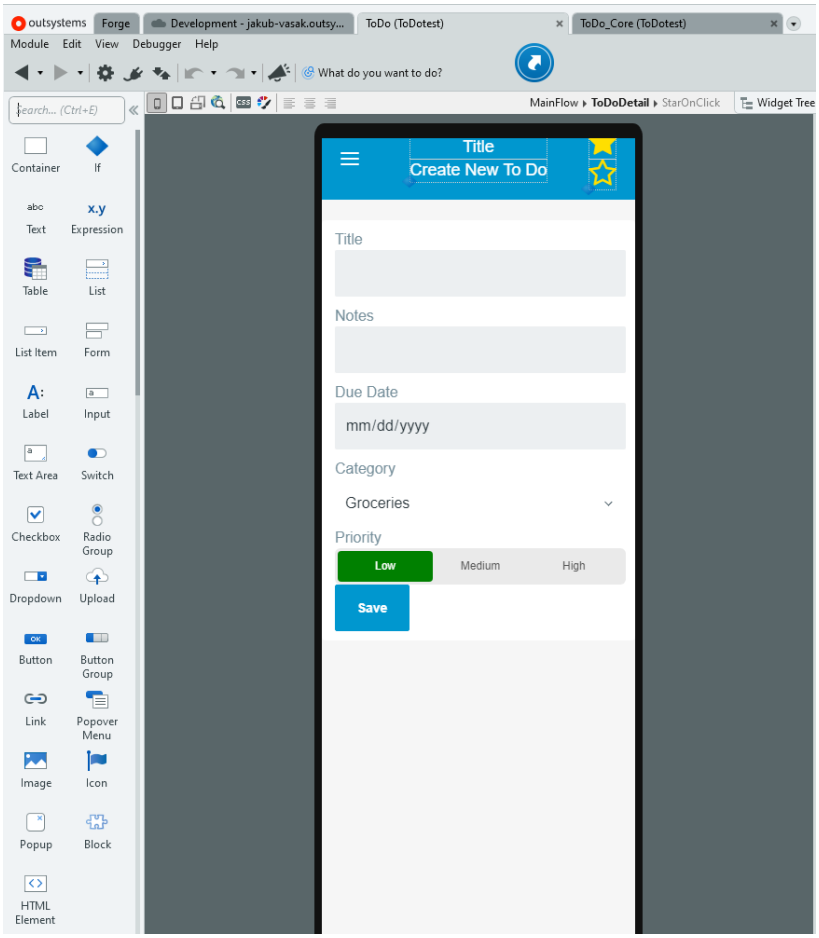
Jak jsem zmínil výše, low-code platformy dovolují uživatelům vytvořit aplikace bez potřeby programování. Hlavní myšlenka konceptu je založena na vytvoření aplikace, která se bude skládat z bloků, které jsou přeprogramovány samotnými programátory této

platformy. Jako příklad bloků zde mohu uvést pole pro psaní, seznamy, formuláře nebo tlačítka. Jednotlivé bloky jsou vždy nastavitelné pomocí jejich vlastností, které si mohou uživatelé upravovat dle své potřeby. Následně mohou být bloky být napojené k entitám v databázích. Zkušení uživatelé si mohou naprogramovat i bloky vlastní, pokud nenalezli řešení v předdefinovaných blocích. Platformy dále nabízí databáze se zjednodušeným vytvářením konkrétních tabulek popisující jejich entitu. Uživatelé mají vždy možnost přidávat další nové záznamy a entity, takže databáze se mohou zvětšovat s potřebami firmy. Díky tomuto konceptu může být jednoduchá aplikace vytvořena relativně rychle. Při vytváření složitějších aplikací platformy nabízí editory toků, které dokážou regulovat datové toky.

Samotný vývoj aplikací v low-code platformách probíhá vytvářením stránek (obrazovek) a následným přetahováním jednotlivých bloků na uživatelem určené místo. Tímto způsobem je zajištěno uživatelsky přívětivé programování aplikací a není zapotřebí žádného specializovaného programátora, který by vytvářel požadované aplikace. Platformy většinou nabízí návody nebo kurzy zdarma, takže lze uživatele jednoduše vyškolit pro tvorbu jak jednoduchých, tak i složitějších aplikací, místo najímání a placení drahých programátorů. Pokud je potřeba rozšířit aplikace, lze vytvořit nové stránky s novými bloky nebo přidat entity do databázového modelu. Low-code platformy nabízejí své vývojové prostředí buď ve formě webových nebo desktopových aplikací. Některé společnosti dokonce nabízí i obě řešení. Všechna provedená práce ve vývojových prostředí se synchronizuje s cloudem, takže veškerá data mohou být dostupná všem vývojářům ve všech prostředí.

Velká výhoda low-code vývojových platforem je přenositelnost vytvořených aplikací mezi systémy. Uživatelům stačí pouze vytvořit jednu aplikaci, která následně může být spuštěna na webu, iOS nebo Androidu. Díky tomuto řešení se mohou ušetřit prostředky za vývoj jedné aplikace na tři různé platformy.

Za nevýhodu je možné považovat uzavřenost systémů. Pokud se firma rozhodne vytvářet aplikace pomocí jedné z low-code platforem, může být velice obtížné přejít na jinou platformu nebo na jiný typ vývoje aplikací. Platformy jsou většinou uzavřené, tudíž není žádná možnost, jak získat zdrojový kód vytvořených aplikace. Proto je vybrání správné platformy, která vyhovuje všem potřebám podniku, velmi důležité a bráno jako klíčový faktor při rozhodování.



Obrázek 4.1: OutSystems – Příklad bloků

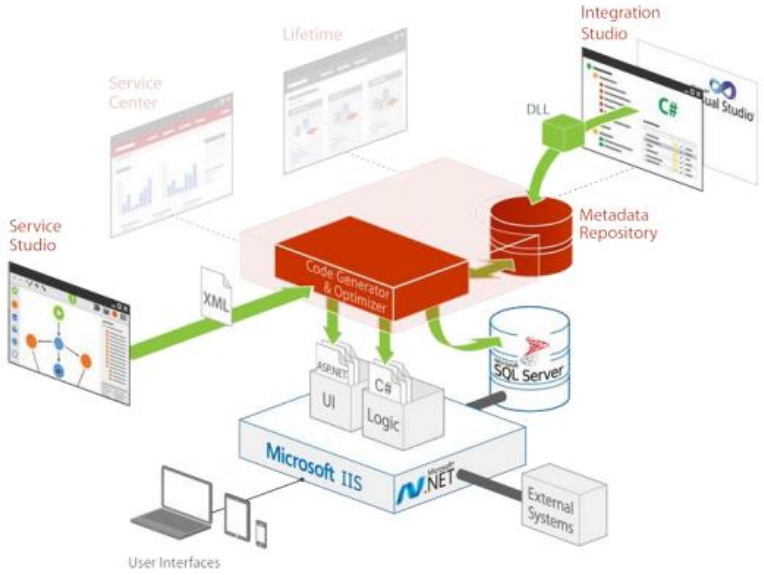
4.1 OutSystems

OutSystems je jedna z nejstarších low-code společností, založena již v roce 2001 v Lisabonu, Portugalsku. Jedná se o otevřenou platformu se standardizovaným kódem, což znamená, že je založena na otevřených standardech a je možnost tuto low-code platformu připojit k jakékoliv databázi nebo systému. Vizuální modely lze rozšířit o standardní nebo vlastní kód, jako je JavaScript, Java, C#, SQL, CSS a HTML. Vývojáři mohou integrovat své aplikace s externími databázemi a stávajícími systémy – dokonce i s nástroji pro testování a monitorování. OutSystems také udržuje uložiště s open-source rozšířeními. Tato rozšíření zahrnují veřejné knihovny nebo pluginy vytvořené pomocí .NET, Java nebo JavaScript s open-source zdrojovým kódem, které lze použít v celé aplikaci.

Platforma OutSystems nepoužívá ke spuštění aplikací vlastní runtime moduly. To znamená, že vývojáři mohou spouštět a měnit nebo aktualizovat své aplikace na jiných platformách nebo pomocí jiných nástrojů.

Všechny aplikace se spoléhají na standardní .NET a Java. Platforma OutSystems se stará o všechny kroky potřebné ke generování, optimalizaci, kompilaci a nasazení aplikací na standardní webový aplikační server, jako je IIS.

Platforma OutSystems nabízí vývoj aplikací zdarma pomocí jejich Free Edice. Po registraci je uživateli link na stáhnutí vývojové platformy, OutSystems nenabízí webové vývojové prostředí. Po nainstalování je uživateli poskytnut interaktivní návod, kde jsou ukázány hlavní objekty a funkce platformy. Tutoriál názorně ukáže uživateli, jak vytvořit svojí první aplikaci, kterou následně může rovnou i vyzkoušet. Po dokončení tutoriálu je uživatel přesměrován na webovou stránku OutSystems, kde mu jsou doporučeny další návody zdarma, rozdělené do několika obtížností a sekcí. O kurzech platformy OutSystems bude více řečeno později.



Obrázek 4.2: OutSystems – Nasazení aplikace na server (OutSystems)

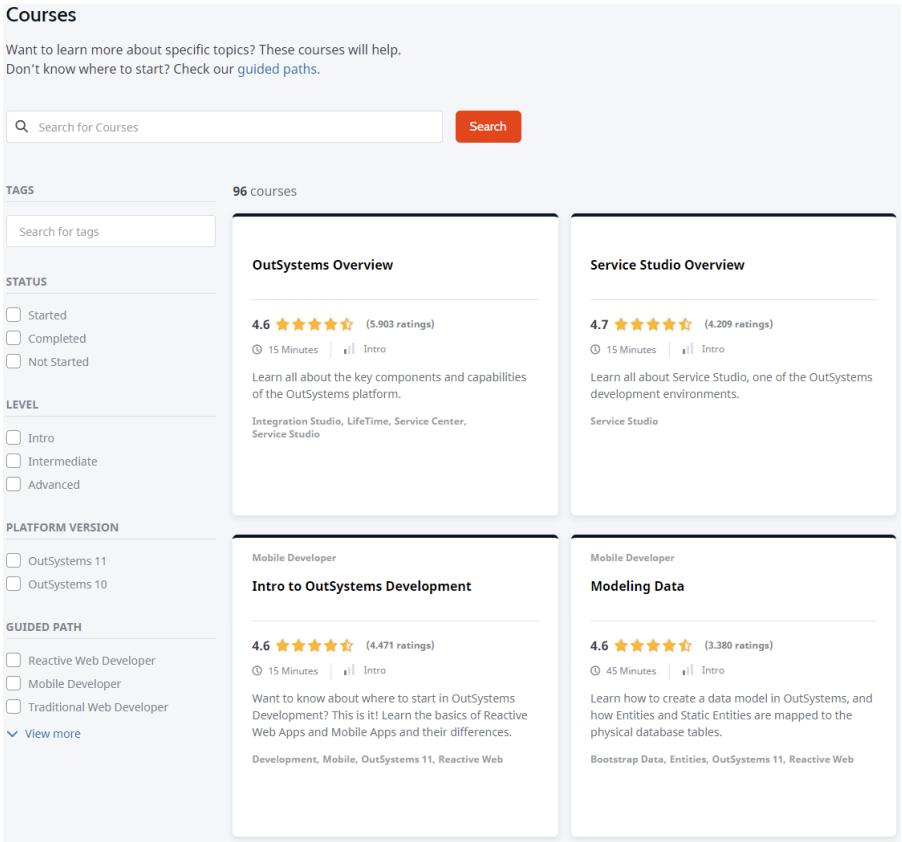
Krom Free Edice nabízí OutSystems další 3 placené edice – Basic, Standard a Enterprise. Edice se kromě ceny liší nabízenými službami a počtem externích koncových uživatelů. OutSystems identifikuje dva různé typy koncových uživatelů:

- Interní uživatelé – jednotlivci zaměstnaní ve firmě, kteří jsou registrovaní jako koncový uživatelé vytvořených aplikací
- Externí uživatelé – jednotlivci nezaměstnaní ve firmě, kteří jsou registrovaní jako koncový uživatelé vytvořených aplikací

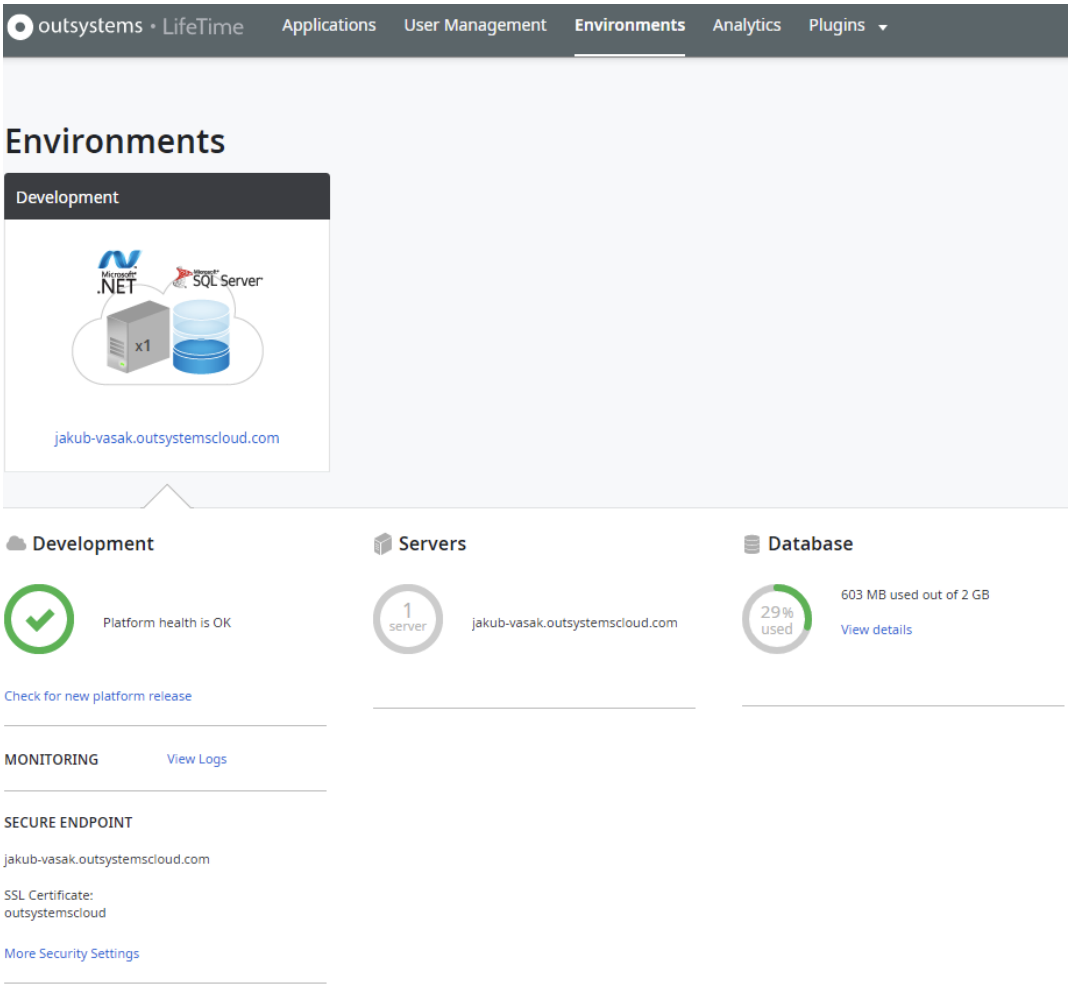
Koncoví uživatelé se prodávají v baleních po 100 pro interní uživatele a 10 000 pro externí uživatele. Cenový systém OutSystems je tedy ideální pro spotřebitelské aplikace s velmi vysokým počtem externích uživatelů, přičemž cena za externího uživatele je 1/100 oproti ceně interního uživatele. Všechny edice platformy zahrnují v základní ceně 100 interních uživatelů. Basic edice začíná na ceně US$4 000 za měsíc a je zaměřená na střední firmy. Standart edice začíná na ceně US$10 000 a cena edice Enterprise se určuje individuálně na základě potřeb zákazníka. Díky tomuto systému může zákazník odejít od platformy OutSystems vždy na konci měsíce (OutSystems).

OutSystems se zaměřuje hlavně na vývoj podnikových aplikací zákazníkům. Působí primárně v Evropě a Severní Americe a jejich zákazníci jsou většinou velké podniky napříč organizacemi zaměřující se na služby, produkty nebo veřejný sektor (Vincent, a další, 2020).

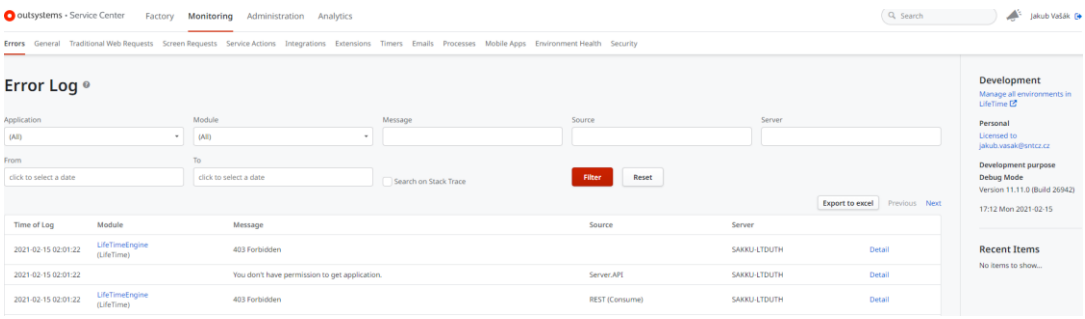
Platforma nabízí vývojářům a správcům prostřednictvím portálů LifeTime a Service Center možnost spravovat a kontrolovat aplikace, monitorovat protokoly, přidělovat uživatelům oprávnění nebo provádět aktualizace aplikací.



Obrázek 4.3: OutSystems – Kurzy



Obrázek 4.4: OutSystems – portál LifeTime



Obrázek 4.5: OutSystems – portál Service Center

4.1.1 Výhody

OutSystems je jedna z nejstarších a nejvyspělejších platforem. Díky tomu má tato platforma největší počet hotových pluginů a šablon připravené pro aplikace. Pluginy

i šablony jsou vytvořeny jak přímo vývojáři platformy, tak komunitou a dají se stáhnout z OutSystems Forge. Navíc Free Edice OutSystems nabízí velkou škálu možností a služeb.

Rozhraní OutSystems nabízí několik předdefinovaných šablon a vzorů uživatelského rozhraní. Uživatelé si také mohou od základu vytvořit vlastní vzory uživatelského rozhraní, které mohou v budoucnu znova využít.

Jak je uvedeno ve zprávě společnosti Gartner „Všechny aplikace vyvinuty citizen-developers jsou robustní a s postupně zvyšující se složitostí aplikace mohou být následně bez problémů vyvíjeny profesionálním vývojářem. Vizuální low-code full-stack jazyk pomáhá usnadnit přenos znalostí a podporuje spolupráci mezi netechnickými a profesionálními vývojářskými týmy“ (Vincent, a další, 2020).

Jak již bylo zmíněno, jedná se o open-source platformu. Aplikace mohou tedy pracovat s různými službami a zdroji dat, které pochází z místních nebo cloudových služeb třetích stran.

4.1.2 Nevýhody

OutSystems má tendenci být upřednostňován spíše profesionálními vývojáři než citizen-developers, zatímco většina ostatních low-code platforem má vyrovnanější podíl vývojářů. Z toho důvodu se může zdát, že tato platforma je o něco náročnější na naučení od konkurence. Další nevýhodou je nutná instalace desktopová aplikace a absence webového vývojového prostředí.

Za největší nevýhodu této platformy se však považuje její vysoká cena, která odrazuje potencionální zákazníky z menších firem.

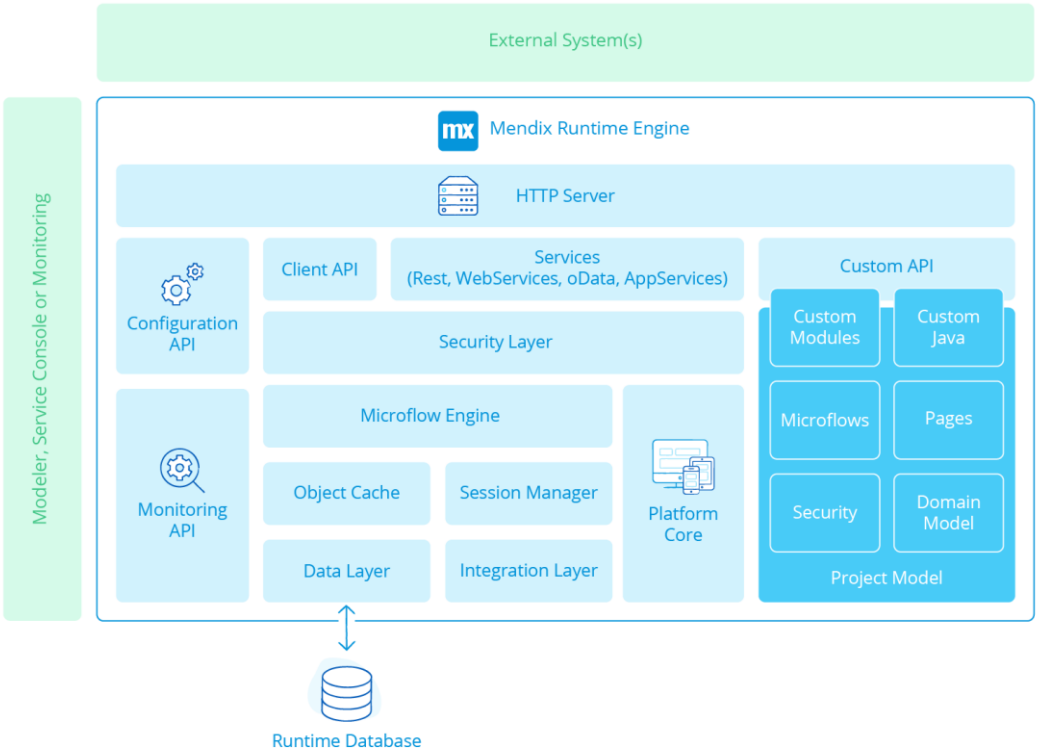
4.2 Mendix

Společnost Mendix byla založena v roce 2005 v Rotterdamu, Nizozemsku. V roce 2018, byla platforma odkoupena společností Siemens. Díky tomu se jí podařilo uzavřít partnerství se společnostmi SAP a IBM.

Mendix nabízí možnosti low-codu i no-codu na jedné plně integrované platformě:

- No-code – Mendix poskytuje webové studio pro čistě vizuální modelování aplikací, které je šité na míru odborníkům v oboru podnikání. Více o no-codu bude probráno později.
- Low-code – k dispozici je také rozsáhlejší desktopové studio, které je určené spíše pokročilejším vývojářům a může být spojeno s ručním psaním kódu pro úpravu vlastností

Mendix je zcela integrovaná aplikační platforma (platform-as-a-service). Platforma je přístupná vývojářům a správcům prostřednictvím portálu pro vývojáře, který poskytuje přístup k aplikacím i službám pro správu požadavků a správě aplikací a aplikačních služeb. Krom vývojových prostředí zde najdeme i Mendix Marketplace, který obsahuje stovky veřejně dostupných pluginů a šablon pro urychlení vývoje. Mendix Marketplace lze nakonfigurovat i pro soukromé použití, takže aplikace a pluginy lze sdílet pouze v organizaci.



Obrázek 4.6: Mendix – Architektura platformy (Mendix)

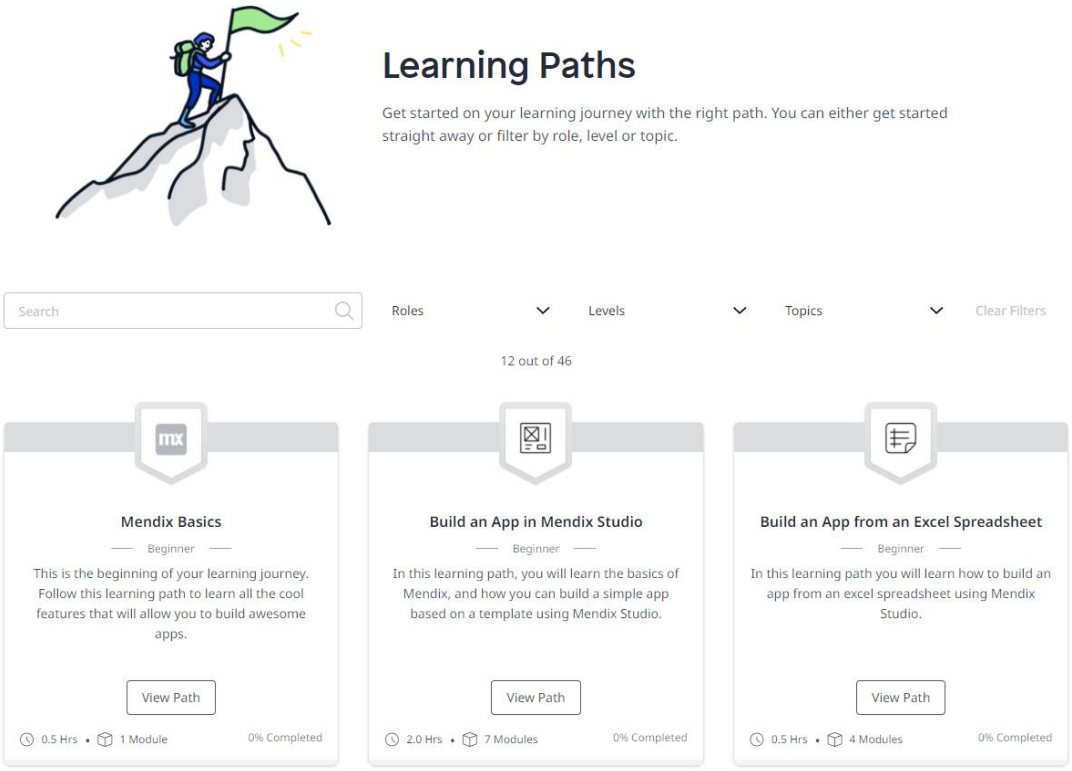
Jak již bylo uvedeno, tak Mendix poskytuje zdarma vývoj aplikací jak z webového prostředí, tak pomocí jejich desktopové aplikace, která navíc umožňuje upravovat aplikace pomocí JavaScriptu a Java. Na rozdíl od OutSystems, je po registraci uživateli poskytnut návod, kde jsou ukázány hlavní objekty a funkce platformy, které si může uživatel hned vyzkoušet ve webovém vývojovém prostředí. Návod názorně ukáže uživateli, jak vytvořit svoji první aplikaci a následně doporučí účast na dalších zdarma návodech, které jsou k nalezení stránkách Mendixu v několika obtížnostech. Jak je možné vidět na obrázku 4.7, návody jsou obsaženy v tzv. „Learning Paths“. Learning Paths jsou následně rozděleny do bloků podle rozsahu zájmu. V učebním postupu Mendix Basics mohou uživatelé získat úvod do základního používání platformy jen za půl hodiny. Každý koncept je vysvětlen v krátkém videu, takže uživatelé mohou vidět využití probírané látky a tu následně sami vyzkoušet.

Mendix dále nabízí další 3 placené edice – Single App, Pro a Enterprise. Jediný rozdíl mezi edicemi Mendix Single App a Pro je, že Single App poskytuje možnost vytvoření pouze jedné aplikace, zatím Pro poskytuje neomezený počet aplikací. Edice Enterprise je rozšířená o další služby a podporu. Jako OutSystems, Medix také rozlišuje 2 typy uživatelů:

- Interní uživatelé – jednotlivci zaměstnaní ve firmě
- Externí uživatelé – jednotlivci nezaměstnaní ve firmě, která mají povolený

Všechny edice začínají s limitem 50 interních uživatelů. Limit externích uživatelů je určen převodovým poměrem mezi externími a interními uživateli 10:1 (maximálně tedy 500 externích uživatelů). Pro rozšíření externích uživatelů je zapotřebí koupit více licencí pro interní uživatele. Cena Single App edice začíná na US$1 918 za měsíc. Tato cena je však odvozena od tříročního závazku, který je potřeba podepsat. V rámci závazku je možno vylepšit edici na verzi Pro nebo Enterprise. Cena Pro edice začíná US$5 418 za měsíc a cena edice Enterprise se určuje individuálně na základě potřeb zákazníka (Mendix).

V nedávné aktualizaci byla přidána Mendix Data Hub pro low-code integraci a dvě možnosti pro privátní cloudové služby – Medix Cloud Dedicated a Medix for Private Cloud. Společnost operuje hlavně v Severní Americe a Evropě (Vincent, a další, 2020).



Obrázek 4.7: Mendix – Kurzy

4.2.1 Výhody

Společnost Siemens pomohla platformě Mendix získat nová partnerství se společnostmi SAP a IBM. Na základě toho Mendix poskytuje specifické možnosti integrace do jakéhokoli systému SAP a uživatelé s účtem IBM Cloud mají přístup k mnoha prostředkům IBM (Mendix).

Za velkou výhodu se dá považovat funkce Mendix Assist, která ve své druhé generaci používá machine learning k integraci kontextových návrhů dalšího kroku v reálném čase v rámci produktu s podporou AI. Nedávno vydaný Mendix Data Hub umožňuje vývojářům snáze objevovat, sestavovat a organizovat data a obchodní funkce pomocí stejných nástrojů, které Mendix používá k vytváření obchodní logiky a aplikačních datových modelů. (Vincent, a další, 2020)

Jako další výhodu lze uvést, že tato platforma nabízí samostatné vývojové prostředí pro začínající programátory (Mendix Studio), tak i robustnější vývojové prostředí pro profesionální vývojáře (Mendix Studio Pro). A v poslední řadě má Mendix velice dobře zpracované učební materiály, což z ní dělá velice vhodnou platformu pro začátečníky.

4.2.2 Nevýhody

Mendix samozřejmě obsahuje i několik slabin. Platforma trochu zaostává ve službách pro správu obsahu v aplikacích. Někteří zákazníci nahlásili, že musí kódovat více než u konkurence, aby splnili integrační potřeby. V posledním roce se Mendix více zaměřil na segment velkých podniků a přešel na novou cenovou politiku, která se cenově přiblížila ostatním předním low-code platformám. Navíc při zakoupení platformy je potřeba podepsat závazek na 3 roky. Pokud bude chtít zákazník odejít během těchto 3 let, je zapotřebí zaplatit poplatek.

4.3 Rozdíl mezi Low-codem a No-codem

Na první pohled je snadné zaměnit low-code a no-code. Zdá se, že i velké analytické firmy je těžko rozlišují. V nejnovějším Magic Quadrantu pro podnikové Low-code aplikační platformy 2020 (Obrázek 3.1) Gartner uvedl, že žádné no-code platformy nebyly zahrnuty. Můžeme zde však nalézt platformu Betty Blocks, u které jejich prodejci tvrdí, že je no-code. V grafu ale byla vyhodnocena jako low-code application platform.

Existují doslova stovky malých detailů a funkcí, které odlišují low-code platformy od no-code řešení. Většina z nich není patrná na úrovni uživatelského rozhraní, odkud pochází velká část zmatku mezi nimi. Ale jak napovídají jejich názvy, hlavní rozdíl mezi low-codem a no-codem je následující:

- Vývoj pomocí low-codu vyžaduje pro úspěšné vytvoření aplikace minimum dovedností v programování – buď od samotných uživatelů nebo s podporou zkušených programátorů.
- Vývoj pomocí no-codu je ještě abstraktnější a uživatelsky přívětivější než low-code vývoj a nabízí uživatelský zážitek zcela založený na vizuálním vývoji, který nevyžaduje vůbec žádné znalosti programování. Pro úspěšné využívání no-code nástrojů by však měl být vývojář stále logicky založený a být schopen konceptualizovat aplikaci od začátku do konce.

Low-code i no-code platformy jsou postaveny na stejné věci, a tou je rychlost. Low-code je vhodný pro vývoj samostatných mobilních a webových aplikací a portálů, které pravděpodobně vyžadují integraci s jinými systémy a několika zdroji dat. Ve skutečnosti jej

lze použít téměř pro cokoli, kromě vysoce sofistikovaných a kritických systémů, které se integrují s více back-endy a externími zdroji dat. No-code by naopak měl být použit pouze pro front-end případy použití. Ačkoli low-code není tak jednoduchý jako no-code, umožňuje mnohem větší diverzitu. (Parker, 2020)

4.4 Srovnání s tradičním vývojem

Měření celkové hodnoty low-code platformy je pro firmu jedna z nejdůležitějších věcí. Pokud by měla firma změnit způsob vytváření softwaru, určitě bude chtít mít představu o tom, jaká je návratnost investice do low-code platformy. I když je jednoduché si představit low-code koncept, může být relativně obtížné jej vypočítat. Proto bych rád zde uvedl příklad ze stránek Mendixu, jak lze pomocí low-code vývoje ušetřit. „Vývoj pomocí low-codu je čtyřikrát rychlejší než tradiční aplikační vývoj. To znamená, že aplikace se může na trh dostat mnohem dříve. Řekněme, že aplikace přinese 200 tisíc v příjmech za měsíc. Tradiční vývoj a následné uvedení do provozu může trvat čtyři měsíce. Pokud se pracuje v kalendářním roce a vývoj začal v lednu, lze tedy získat v příjmech hodnotu za 8 měsíců (1,6 mil.). Vývoj v Mendixu znamená, že aplikace může vyjít místo čtyř měsíců za pouhý jeden měsíc. Tím se získají další tři měsíce, kdy aplikace získává tržby (2,2 mil. za celý rok).“ (van Oosten, 2020).

Tradiční nebo na zakázku vyrobený software je poměrně nákladný. Dle firmy Soltech, která specializuje na vývoj mobilních a webových aplikací, se takové náklady můžou se pohybovat kdekoli v rozmezí 40 000 - 250 000 US$ za návrh a vývoj aplikace. Ačkoli se náklady budou lišit v závislosti na rozsahu projektu a individuálních potřebách společnosti, stále jsou značně vysoké. Toto je obvykle vhodná volba pro velké společnosti, které potřebují vysoce přizpůsobený software, který je obtížné vytvořit pomocí low-code řešení (Mooney).

Low-code platformy jsou ve srovnání s jejich tradičními protějšky mnohem levnější. Je to především proto, že u low-code platforem platí společnost pouze za přístup k zadané službě, nikoli za její vývoj od začátku do konce. Navíc platformy, které jsou aPaaS (aplikační platforma jako služba), jsou obvykle provozovány a udržovány společností, která platformu vlastní. Díky tomu je proces údržby aplikací jednodušší, protože všechny aktualizace a vylepšení softwaru zpracovává hostitelská společnost.

Obrázek 4.8: Mendix – příklad vyhodnocení hodnoty low-code platformy (van Oosten, 2020)

4.5 Souhrn

Za posledních 20 let raketově vzrostla rozmanitost technologií, které mají organizace k dispozici. Výsledkem je, že IT oddělení již nespravují homogenní aplikace, ale různorodá a složitá prostředí. A jak se tato prostředí vyvíjejí, vyvíjí se i IT.

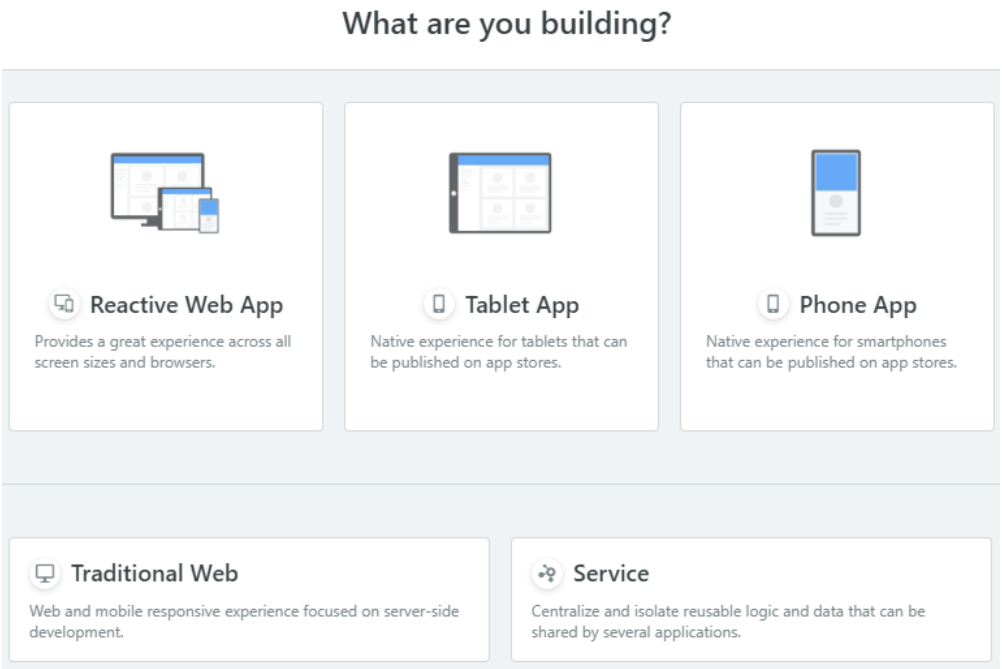
Spoléhání se na vlastně napsané skripty při vývoji softwaru již dnes neodpovídá rychlosti a hbitosti, které podniky vyžadují, a navíc specializace vývojářů na konkrétní platformu je v rozporu s požadavky cross-platform (software, který je implementován na několika platformách), které podniky stále více vyžadují. Velice málo vývojářů má zkušenosti se všemi nástroji pro vývoj softwaru a je velice drahé si najímat několik programátorů pro vytvoření jedné aplikace na několik platforem. A právě zde přichází na řadu low-code platformy, které poskytují drag-and-drop rozhraní umožňující rychle

sestavovat nové procesy a vytvářet aplikace, aniž by museli programátoři zkoumat, psát a testovat nové skripty. Stejně důležité je, že programátoři nemusí mít specializované znalosti pro každou platformu a není potřeba vytvářet separátní aplikace pro Android a iOS. Navíc se správným nástrojem může téměř každý člen IT týmu vyvinout spolehlivé mezi-platformní procesy, a tak zkrátit čas k dodání aplikací. Předpřipravené funkční bloky, které mohou být použity bez hlubší znalosti programování, dokážou bezpochybně ušetřit čas během vývoje. Vývojář se může zaměřit na aplikační logiku namísto vytváření komunikací s databází, získávání potřebných dat nebo vytváření front-end aplikací. Nicméně vývoj speciálních aplikací, které potřebují velmi upravené funkcionality a integraci s několika systémy, může být na low-code platformě velice náročný. Pro takové aplikace nemusí být low-code tím pravým řešením.

5 Tvorba vzorové aplikace

5.1 Návrh aplikace

Tato část se bude zabývat tvorbou jednoduché aplikace pro inventarizaci zařízení pro firmu S&T. Za low-code vývojové prostředí byla vybrána platforma OutSystems, se kterou mám nejvíce zkušeností. Aplikace bude tvořena pomocí desktop vývojového prostředí OutSystems Service Studio 11 a bude směřována především na mobilní platformu Android. Finální verzi aplikace je možné spravovat přes portál OutSystems LifeTime. Navíc tento portál umožňuje i aplikaci otestovat přímo v prohlížeči.



Obrázek 5.1: OutSytems – možnost zaměření aplikace

Aplikace se bude skládat z tzv. modulů. Moduly umožňují strukturovat aplikaci do několika částí, přičemž každá část slouží ke konkrétnímu účelu. Ukázková aplikace je rozdělená na dva moduly. První a hlavní modul (SnT) obsahuje User Interface aplikace a druhý modul (SnTCore) slouží jako datový model. Entity datového modulu musí být nastaveny jako veřejné, aby mohl je modul SnT využít. Výhodou tohoto přístupu je, že veškerá data, která se uloží do databáze, se budou moci využít i v jiných aplikacích. Této vlastnosti se využije pro exportování a stažení výsledné databáze do excelovského souboru.

5.1.1 Základní požadavky na vyvíjenou aplikaci

Hlavním účelem aplikace je zefektivnění práce a vyloučení lidského faktoru při instalacích nových zařízení a odvozu starých zařízení z Komerční banky.

Popis funkčnosti:

- Samotná aplikace bude tvořena ze čtyř obrazovek
- Na první obrazovce budou uvedena čísla instalačních protokolů, které má technik k dispozici. Tyto protokoly budou naimportovány z excelovského souboru s testovými daty. Pokud technik nenajde požadovaný protokol, bude mít možnost přidat nový záznam na obrazovce číslo 2.
- Po rozkliknutí čísla požadovaného protokolu se přejde na třetí obrazovku, kde bude zobrazen list existujících záznamů sériových čísel hardwaru. Následně má technik možnost přidat nový záznam sériového číslo hardwaru nebo upravit existující záznam.
- Čtvrtá obrazovka bude obsahovat 4 povinná textová pole a 1 list.
    - Do 1. pole se budou skenovat sériová čísla HW. Pokud bude sériové číslo nečitelné, bude možnost ho dopsat ručně.
    - Do 2. pole se bude skenovat unikátní kód banky, aby se rozlišilo, z jaké pobočky je HW odvážen.
    - 3. pole bude sloužit k ručnímu doplnění modelu HW technikem
    - 4. pole bude sloužit k vyplnění data instalace/odvezu HW. Automaticky se zde vyplní aktuální datum.
    - List bude sloužit k výběru dvou možností – zda se jedná o instalovaný HW nebo odvoz starého HW
- Veškeré uložené záznamy budou editovatelné v případě chybného zadání technikem
- Výsledná databáze se bude dát exportovat do excelovského souboru
- Aplikace bude pro platformu Android

Před začátkem vývoje bylo nutné projít několika kurzy, které jsou součástí tzv. „Becoming a Mobile Developer Guided Path“. Jak již bylo zmíněno, aplikace byla vytvořena pouze v jedné verzi, a to pro mobilní platformu Android. Přenesení na platformu iOS může být provedeno ve vlastnostech projektu, tato možnost však nebyl vyzkoušena.

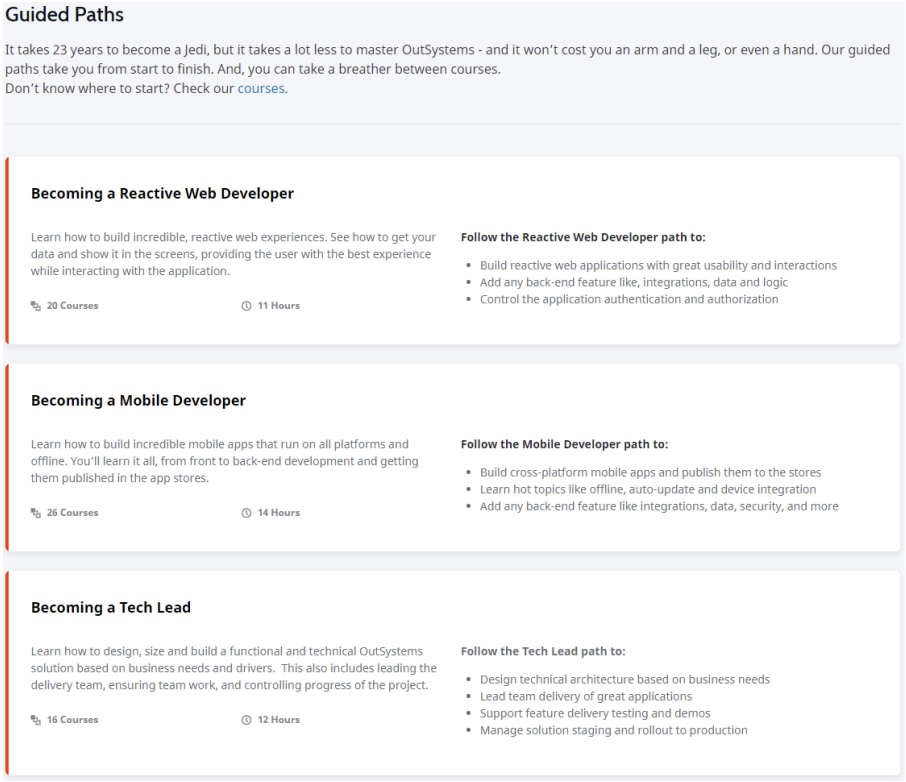
Posledním požadavkem bylo vytvoření aplikace do časového limitu 35 hodin včetně studia materiálů.

5.2 Návody

OutSystems poskytuje uživatelům komplexní výukové programy. Existuje několik způsobů, jak se naučit základní nebo pokročilejší funkce. První možností jsou kurzy. Kurzy lze najít samostatně (obrázek 4.3) nebo jsou součástí tzv. „Guided Paths“. Jak lze vidět na obrázku 5.2, jednotlivé „cesty“ jsou rozděleny do bloků podle rozsahu zájmu. Jak již bylo zmíněno výše, pro tento projekt byla vybrána cesta Becoming a Mobile Developer. Podle popisu této cesty zde uživatele dostanou úvod do základního používání platformy a následně se naučí, jak vytvářet mobilní aplikace, od front až k back-end vývoji. Konec této cesty je věnovaný publikování aplikací v obchodech s aplikacemi pro jednotlivé platformy. Becoming a Mobile Developer je složen 26 kurzů a doručený strávený čas je 14 hodin. Kurzy jsou uspořádány tak, aby na sebe logicky navazovaly a pro splnění pokročilejších kurzů je potřeba splnit kurzy základní. Jednotlivé kurzy jsou rozděleny na kapitoly, kde se ve videích rozebírá právě řešený koncept. Každá kapitola je ukončena testem, který musí být splněn na 100 %, aby bylo možné pokročit do další kapitoly. Pokud uživatel odpoví na otázku v testu špatně, je pod ní následně vypsáno krátké vysvětlení. Díky tomu uživatel ví, v čem udělal chybu. Každý kurz je ukončen cvičením, kde jsou dostupné materiály, pomocí kterých si můžou uživatelé sami vyzkoušet probranou látku přímo v OutSystems Studiu. Cvičení pouze řeší probíraný problém na cvičném příkladu a příklady na sebe nenavazují. V minulosti však zmíněná cesta obsahovala i materiály, pomocí kterých šlo vytvořit cvičnou aplikaci. Cvičná aplikace pak nabírala na složitosti po každém absolvovaném kurzu. Bohužel tato možnost byla odebrána s aktualizací, která vyšla 1. 1. 2021. Další výukové kurzy lze najít na stránkách OutSystems. Ty již však nejsou součástí Guided Paths a jsou zaměřeny na konkrétní témata pro hlubší vysvětlení problémů.

Dalším způsobem, jak se naučit v platformě OutSystems, je účastí na tzv. „Boot Camps“. Boot Camp je tréninkový kurz zdarma vedený certifikovaným instruktorem. Kurzy trvají několik dní a během nich jsou vytvářeny zkouškové aplikace, na kterých jsou vysvětleny základní nebo pokročilé funkce. Na konci každého kurzu je poskytnuta volitelná certifikační zkouška. Každý Boot Camp se zaměřuje na jinou oblast zájmu. Momentálně lze absolvovat kurzy pouze na dálku. Za finanční poplatek je možnost

vytvořit soukromý kurz speciálně na míru potřebám firmy. Na stránkách OutSystems můžeme najít i online záznamy z přednášek, kde jsou vysvětlena nejrůznější témata týkající se platformy OutSystems. OutSystems dále nabízí rozsáhlé stránky s dokumentací, diskusní stránky s vývojáři a komunitní fórum.



Obrázek 5.2: OutSytems – Guided Paths

5.3 Tvorba datového modelu

Datový model lze vytvořit buď během procesu vývoje v hlavním modulu společně s UI, nebo vytvořit v samostatném modulu. V případě potřeby je možné během procesu vývoje vytvořit novou entitu, která se poté se automaticky přidá do hlavního modulu aplikace. OutSystems automaticky vytváří jedinečné identifikátory pro entity, takže při vytváření entity není nutné je přidávat jako jeden z atributů. Datový model této aplikace byl vytvořen před vývojem aplikace v modulu SnTCore a je sestaven ze tří entit, SodiNumber, Hardware a Category. Tyto entity představují požadavky, které musí technik vyplnit při tvorbě nového záznamu. Entita představuje prvek reálného světa a je popsána svými vlastnostmi, které se považují za atributy. Všechna data z aplikace budou uložena v online databázi, tím pádem aplikace musí být vždy připojena k internetu.

Entita Hardware představuje záznam jednoho zařízení, které technik bude přidávat do datového modelu. Převážně se bude jednat o notebooky, přenosné stolní PC a monitory. Atributy Hardwaru jsou:

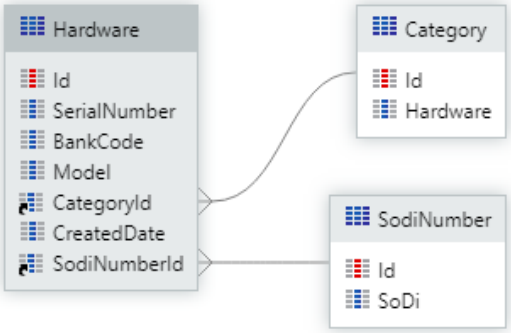
- Id – Tento atribut je automaticky vytvořen platformou OutSystems při vytvoření entity a slouží jako primární klíč, který jednoznačně identifikuje záznam v tabulce. Jako datový typ byl automaticky vybrán Long Integer.
- SerialNumber – Představuje sériové číslo zařízení, které bude naskenováno. Tento atribut bude v databázi považován jako jedinečný a nebude možné přidat dvě stejná sériová čísla. Výsledek skenu má datový typ Text, tudíž je nutné nastavit jako datový typ atributu Text.
- BankCode – Každá pobočka banky má svůj unikátní kód. Tento je kód je zobrazený na štítku, který je následně nalepený na Hardwaru. Pomocí tohoto kódu se rozlišují jednotlivé pobočky banky. Výsledek skenu má datový typ Text, tudíž je nutné nastavit jako datový typ atributu Text.
- Model – Slouží k jednoduchému popsání hardwaru. Datový typ byl nastaven na Text.
- CreatedDate – Čas vytvoření záznamu. Datový typ byl nastaven na Date a výchozí hodnota na „CurrDate()“, která vyplní aktuální den.
- CategoryId – Cizí klíč odkazující na entitu Category.
- SodiNumberId – Cizí klíč odkazující na entitu SodiNumber.

Firma S&T využívá databázový systém SoDi (Solve Direct), který slouží jako nástroj pro správu požadavků, incidentů a servisních zásahů. Požadavky představují SLA (Service Level Agreement) smlouvy uzavřené se zákazníky. Každý požadavek má unikátní číslo SoDi, které zároveň slouží jako číslo protokolu. Entita SodiNumber představuje čísla firemních protokolů. Obsahuje pouze dva atributy, Id a SoDi. SoDi představuje právě název protokolu z firemní databáze. Testové protokoly byly naimportovány z excelovského souboru. Pokud technik nenajde požadovaný protokol, bude mít možnost přidat nový. Atribut SoDi bude nastaven jako jedinečný. Datový typ atributu nastaven na Text.

Entita Category slouží pouze jako list, ve kterém technik vybere, zda hardware na pobočku přijel instalovat nebo naopak z pobočky odvést. Tato entita byla vytvořena pomocí importu z Excelu a obsahuje pouze dva atributy, Id a Hardware. Datový typ atributu Hardware nastaven na Text.

Vazby mezi entitami jsou následující:

- 1:N mezi SodiNumber a Hardware. Hardware může mít přiřazené pouze jedno SodiNumber, ale SodiNumber může obsahovat několik záznamů o hardwaru
- 1:N mezi Category a Hardware. Hardware může mít přiřazenou pouze jednu kategorii, ale pod jednou kategorií může být několik záznamů o hardwaru



Obrázek 5.3: Datový model

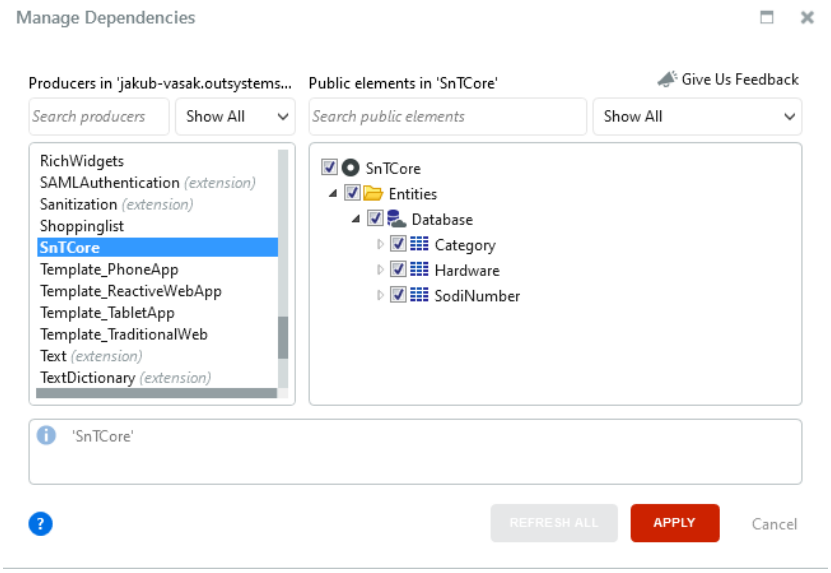
OutSystems umožňuje nastavit jednotlivé atributy jako povinné. Povinné atributy jsou vždy zvýrazněny hvězdičkou. To znemožní technikovi uložení záznamu, pokud nevyplnil povinné atributy. Atribut Id je vždy nastaven jako povinný. Po vzniku jakékoliv entity OutSystems automaticky vytvoří šest akcí, které poskytují typické funkce CRUD – Create, Read, Update a Delete.

5.4 Tvorba grafického rozhraní

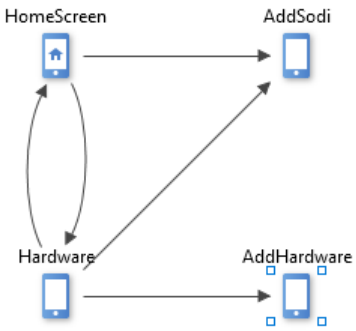
Obrazovky jsou rozhraním pro koncového uživatele vytvořené aplikace. Obrazovka se skládá z bloků, které jsou pro usnadnění práce s nimi předdefinovány. Bloky samozřejmě můžeme vytvářet či upravovat. Každá stránka má své rozložení, ve kterém jsou uspořádány widgety. Existuje několik typů widgetů seskupených do kategorií, jako příklad mohou být uvedeny: menu widgety, datové widgety, widgety pro tlačítka, widgety pro grafy a další. Velkou výhodou OutSystems je automatického upravování velikostí jednotlivých widgetů v závislosti na poloze telefonu nebo tabletu.

Na obrázku 5.6 je uveden příklad prázdné stránky. Na rozvržení vybrané stránky je patrné, že stránka je rozdělena do 3 hlavních částí – záhlaví, hlavní obsah a zápatí. Jak již

bylo uvedeno, grafické rozhraní aplikace a následná logika je obsažena v modulu SnT. Pro použití entit vytvořené v modulu SnTCore je potřeba vytvořit mezi nimi závislost, jak je možno vidět obrázku 5.4. Tyto entity budou následně použity k vytvoření uživatelského rozhraní ve všech čtyřech obrazovkách. Změny v datovém modelu lze provést kdykoli během nasazení aplikace. Mohou však nastat některé chyby, zejména pokud z tabulky jsou mazána data, která obsahují i vazby mezi entitami. Řešením tohoto problému je nastavení ochrany Cizích klíčů. Toto pravidlo nám zabrání smazání záznamu entity v případě, když jiná entita obsahuje data, která na tento záznam odkazuje. Obrazovky jsou propojeny a možná navigace v celé aplikaci je ukázána na obrázku 5.5. V OutSystems je také možné vytvořit různé uživatelské profily, kde lze definovat, jaká data budou pro uživatele k dispozici. Tato možnost však nebyla v projektu využita.



Obrázek 5.4: Přidání závislosti modulu



Obrázek 5.5: Navigace v aplikaci

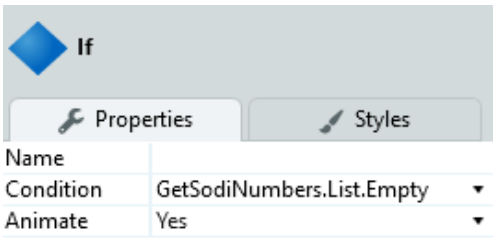
Obrázek 5.6: Rozdělení listu

5.4.1 Hlavní obrazovka

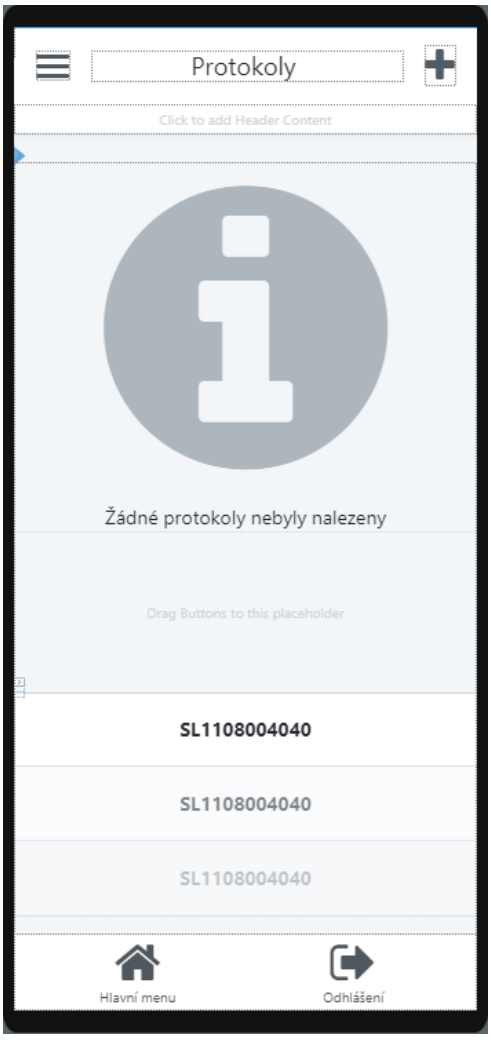
První obrazovka zároveň slouží jako hlavní obrazovka, v níž je uveden seznam firemních protokolů, které má technik k dispozici. Obrazovka se vytváří přetažením položky „Screen“ do pracovní plochy. Následně je potřeba naplnit obrazovku daty z databáze. Jako zdroj byla použita entita SodiNumbers. Tím se vytvoří tzv. Aggregate neboli Agregát s názvem GetSodiNumbers. Ten má přístup ke všem datům z entity SodiNumbers.

Přidáním widgetu „If“ do hlavního obsahu se vytvoří podmínka. Podmínka vyhodnocuje, zda seznam, který byl přidán v dalším kroku, je prázdný nebo ne. V oknu vlastností je položka „Condition“, do které je nutné napsat kód uvedený na obrázku 5.7. Dále je zde vlastnost „Animate“. Tato vlastnost při nastavení na hodnotu „Yes“ provede animaci obsahu jakmile podmínka If změní hodnotu. Pokud podmínka skončí s hodnotou True, stránka zobrazí text „Žádné protokoly nebyly nalezeny“. V opačném případě se zobrazí seznam naplněný daty z agregátu GetSodiNumbers, který obsahuje čísla instalačních protokolů. Seznam se vytvoří přetáhnutím widgetu „List“ do větve podmínky False.

Po rozkliknutí jména protokolu je technik přesměrován na obrazovku Hardware, kde je seznam všech záznamů, které patří pod příslušný protokol. Aby bylo možné zobrazit tento seznam, musí být nové obrazovce předána informace, která obsahuje ID právě vybraného protokolu. V pravém horním se nachází ikona Plus, která po zmáčknutí přesměruje technika na novou obrazovku, kde může vytvořit nový protokol. Na obrázku 5.8 lze vidět, jak tato obrazovka vypadá po naplnění daty.



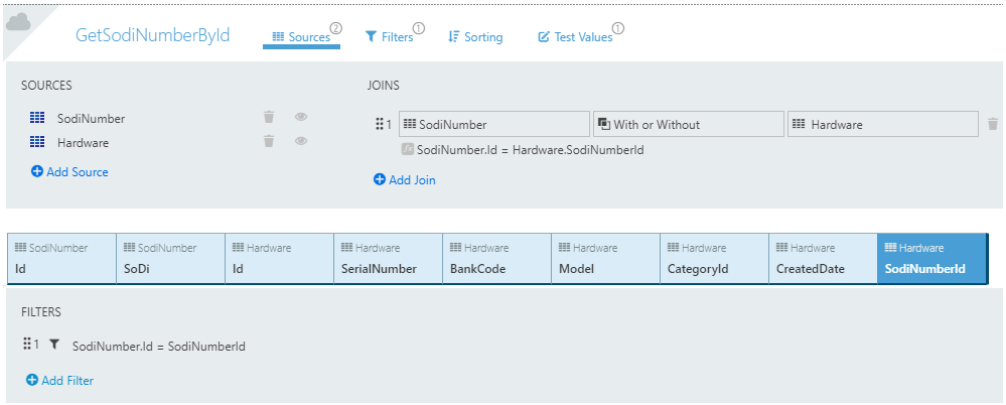
Obrázek 5.7: Podmínka



Obrázek 5.8: Hlavní stránka

5.4.2 Hardware

Hardware obsahuje list se všemi dostupnými záznamy sériových čísel, které jsou uloženy pod daným protokolem. Obrazovka dostává vstupní parametr s ID vybraného protokolu (SodiNumberId). Tento vstupní parametr nám také vytvoří agregát GetSodiNumbersById, který obsahuje data z entity SodiNumber. Následně bylo potřeba do agregátu připojit entitu Hardware. Když je do agregátu přidána druhá entita, automaticky se také vytvoří spojení mezi entitami. Entita může být z agregátu kdykoliv odebrána nebo může být změněno pouze spojení mezi entitami. Na obrázku 5.9 je znázorněno, jak takové spojení vypadá a filtr, který byl agregátu přidán. Ten zaručí, že agregát bude filtrován tak, aby vracel pouze sériová čísla vytvořená pod aktuálním číslem SoDi.



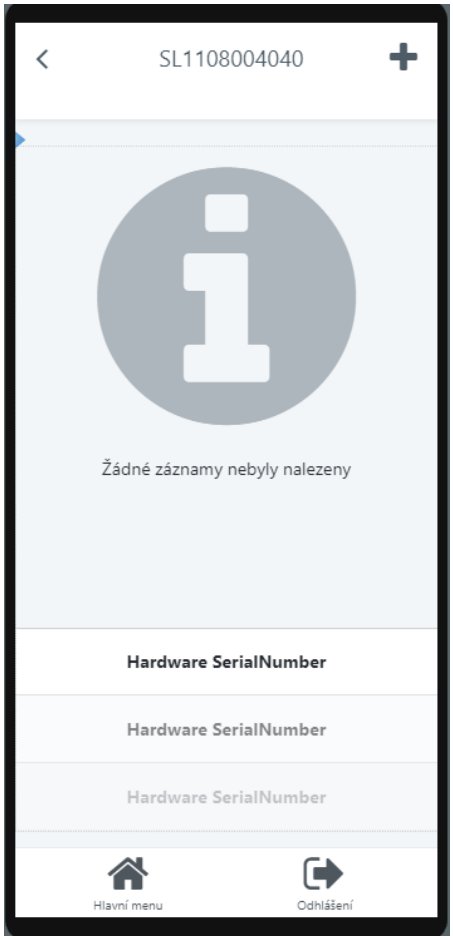
Obrázek 5.9: Agregát SoDiNumberById

Stav vstupní podmínky stránky byl nastaven:

```
GetSodiNumberById.List.Current.Hardware.SerialNumber =
NullTextIdentifier()
```

Výstup podmínky s hodnotou True znamená, že list neobsahuje žádná sériová čísla a stránka zobrazí text „Žádné záznamy nebyly nalezeny“. V opačném případě se zobrazí seznam naplněný daty z agregátu GetSodiNumberById. V levém horním rohu obrazovky se nachází ikona šipky zpět. Tato ikona odkazuje zpět na hlavní obrazovku. Jako název obrazovky se zobrazuje právě vybraný protokol. Po zmáčknutí je možno upravit jméno protokolu. Technik je po rozkliknutí sériového přesměrován na novou obrazovku, kde může upravit existující záznam, který patří pod příslušný protokol. Aby bylo možné záznam upravit, musí být nové obrazovce předána informace, která obsahuje ID právě vybraného

záznamu a protokolu. Po zmáčknutí ikony Plus je technik přesměrován na novou obrazovku pro vytvoření nového záznamu.



Obrázek 5.10: Obrazovka Hardware

5.4.3 AddHardware

AddHardware umožňuje vytvořit nový nebo upravit již existující záznam sériového čísla. Obrazovka dostává dva vstupní parametry, ID vybraného protokolu (SodiNumberId) a ID vybraného sériového čísla (HardwareId). Při vytváření nového záznamu je obrazovce předán NullIdentifier() jako ID sériového čísla. Předáním NullIdentifier() se agregát obrazovky pokusí najít záznam s nulovým ID v databázi. Tato operace vždy skončí neúspěchem, protože v databázi nejsou žádné záznamy bez ID pole (ID je povinné a automatické číslo). Vzhledem k tomu, že není nalezen žádný záznam, budou vstupní pole formuláře prázdná. Technik pak může vyplnit vstupy a kliknutím na tlačítko „Uložit“ odeslat data na server. Vstupní parametr HardwareId nám také vytvoří agregát GetHardwareById, který obsahuje data z entity Hardware.

V názvu obrazovky byla umístěna podmínka, která rozlišuje, zda se vytváří nový nebo upravuje již existující záznam. Stav podmínky byl nastaven:

```
HardwareId <> NullIdentifier()
```

Tato podmínka definujte název obrazovky v závislosti na hodnotě vstupního parametru předaného obrazovce. Pokud je parametr HardwareId roven NullIdentifier(), v názvu obrazovky se zobrazí text „Vytvořit nový záznam“. Pokud má parametr hodnotu, zobrazí se jako název obrazovky hodnota atributu SerialNumber. V levém horním rohu obrazovky se nachází ikona šipky zpět, která odkazuje předcházející obrazovku.

Pomocí „Form widget“ byl vytvořen formulář, který umožní obrazovce přijímat vstupy od koncových uživatelů a vyplňovat informace pro atributy entity Hardware. Vstupy se vytvoří jednoduše přetažením požadovaných atributů do widgetu na obrazovku. Speciálním případem je přetažení atributu CategoryId. Jelikož se jedná o cizí klíč, vznikne rozevírací seznam, který obsahuje všechny hodnoty z entity Category. Po přetažení požadovaných atributů, widget automaticky vytvoří tlačítko „Uložit“. Pod vstupy sériové číslo a kód banky byly přidány tlačítka pro naskenování čárového kódu. Díky tomu by měl být technik schopen naskenovat sériové číslo, kód banky, vyplnit model zařízení, datum provedení práce, vybrat kategorii (v rozevíracím seznamu) a kliknutím na tlačítko uložit data do databáze. Byla přidána možnost i smazat existující požadavky z databáze. Logika, která bude vytvářet nebo upravovat záznamy po stisknutí tlačítek, bude rozebrána později.

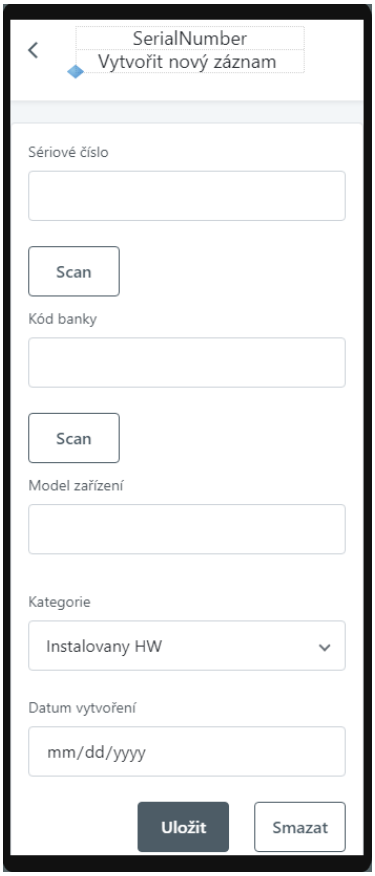
5.4.4 AddSodi

AddSodi umožňuje vytvořit nový nebo upravit již existující protokol. Obrazovka dostává vstupní parametr ID vybraného protokolu (SodiNumberId). Při vytváření nového protokolu je znovu obrazovce předán NullIdentifier() jako ID protokolu. To zajistí, že vstupní pole bude prázdné. Technik pak může vyplnit vstup a kliknutím na tlačítko „Uložit“ odeslat data na server. Vstupní parametr SodiNumberId nám také vytvoří agregát GetSodiNumbersById, který obsahuje data z entity SodiNumber.

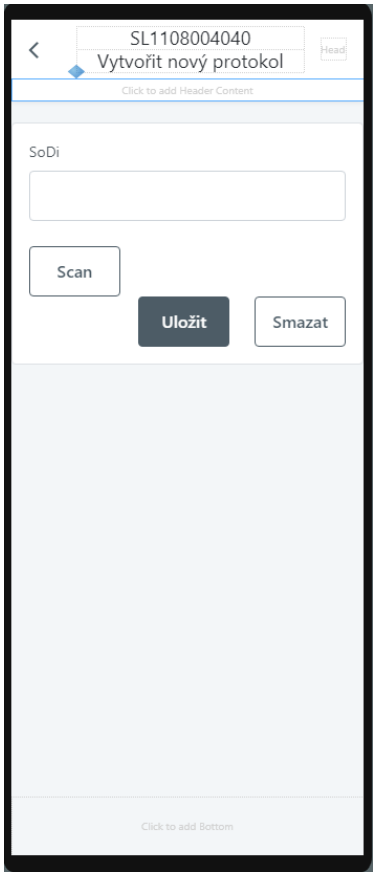
V levém horním rohu obrazovky je ikona šipky zpět odkazující na předcházející obrazovku. Název obrazovky obsahuje stejnou podmínku jako obrazovka AddHardware, která rozlišuje, zda se vytváří nový nebo upravuje již existující protokol. Stav podmínky byl nastaven:

```
SodiNumberId <> NullIdentifier()
```

Formulář tentokrát obsahuje pouze jeden vstup pro atribut entity SodiNumber. Hodnotu pro tento atribut lze naskenovat z čárového kódu. Byla přidána možnost smazat existující prázdné protokoly z databáze.



Obrázek 5.11: Obrazovka AddHardware



Obrázek 5.12: Obrazovka AddSodi

5.5 Tvorba logiky

Akce v OutSystems je prvek, který nám umožňuje definovat logické toky. Jako příklad můžu uvést akce pro vytváření nebo aktualizace objektů běžící buď na straně serveru nebo na straně klienta. Tok akce je místo, kde je přímo definována část logiky. Každý tok má právě jeden začátek, konců může mít však několik. Za konec akce se považuje ukončení činnosti nebo přesměrování na jinou obrazovku. Jinými slovy lze tok akce popsat jako zjednodušený model procesu vizuálně představící logiku, která je potřeba udělat pro úspěšně provedení určité úlohy. OutSystems nabízí 3 druhy akcí:

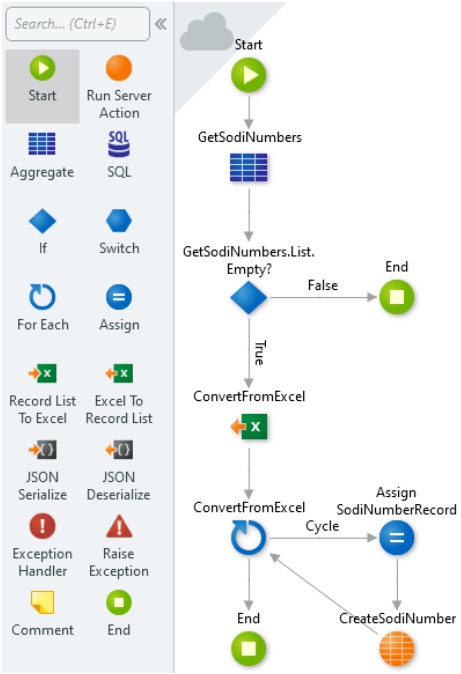
- Screen Actions – logika, která je specifická pouze pro jednu obrazovku

- Client Actions – logika, která může být využita kdekoliv na koncovém zařízení
- Server Actions – logika, která probíhá na straně serveru

V aplikaci bylo využito několik Screen Actions, dvě Server Actions a dvě Client Actions.

Akce Naplnění databáze

Tato serverová akce naplní databázi testovými daty SoDi ze souboru aplikace Excel za předpokladu, že je odpovídající entita SodiNumber stále prázdná. Funkce ConvertFromExcel nalezne sloupec s názvem SoDi a konvertuje všechny hodnoty v daném sloupci do databáze. Přidáním cyklu For Each a funkce Assign tato akce přiřadí všechny hodnoty z excelu k atributu SoDi a vytvoří nový záznam. Na obrázku 5.13 lze nalézt finální podobu této akce. Stejná akce se využívá i pro naplnění daty entitu Category.



Obrázek 5.13: Server Action

Akce Uložit

Jedná se o akce, které jsou specifické pouze pro obrazovky AddHardware a AddSodi. Je zde naimplementována logika skrývající se pod tlačítky Uložit, které vytváří nebo aktualizují data v databázi. Jako příklad bude popsána akce Uložit na obrazovce AddHardware.

Jak již bylo zmíněno, při vzniku jakékoliv entity OutSystems automaticky vytvoří šest akcí. Na obrázku 5.14 je možné vidět použití akce CreateOrUpdateHardware. Tato akce přidá nový záznam entity do databáze. Pokud záznam již existuje, budou všechny atributy nahrazeny novými. Nejdříve je však nutné pomocí podmínek ověřit správnost vstupů a vytvořit chybové hlášky, které se zobrazí při nesplnění požadovaných kritérií. První tři podmínky kontrolují délku vstupů zadaných technikem. Při neodpovídající délce vstupu lze pomocí funkce Assign vypsat chybovou hlášku požadující určitý minimální počet znaků u příslušného vstupu. Čtvrtá podmínka kontroluje, zda vstupní parametr HardwareId má již vytvořené ID:

```
HardwareId <> NullIdentifier()
```

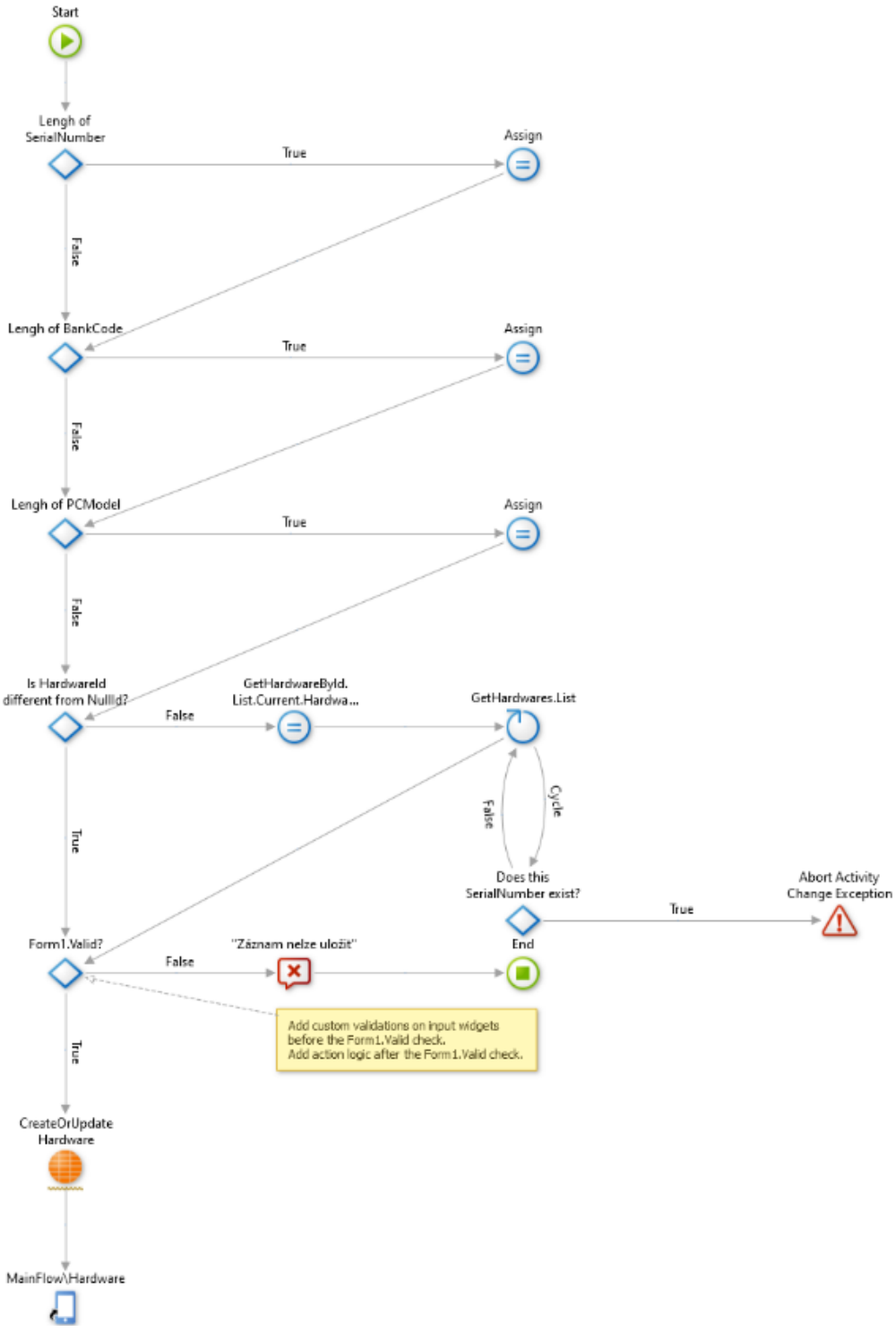
Pokud podmínka skončí s hodnotou True, tak HardwareId nemá nulové ID a jedná se tedy úpravu již existujícího záznamu. Pokud naopak má HardwareId nulové ID, podmínka skončí s hodnotou False a do atributu SodiNumberId se přiřadí aktuální ID protokolu. Následuje cyklus For Each s podmínkou kontrolující, zda stejná hodnota atributu SerialNumber již neexistuje v databázi. Pokud podmínka skončí pravdivě, aplikace vypíše chybovou hlášku „Nelze přidat záznamy se stejnými sériovými čísly“. V opačném případě aplikace zkontroluje validaci vstupů a vytvoří nebo aktualizuje záznam. Akce je ukončená přesunem na obrazovku Hardware.

Akce Smazat

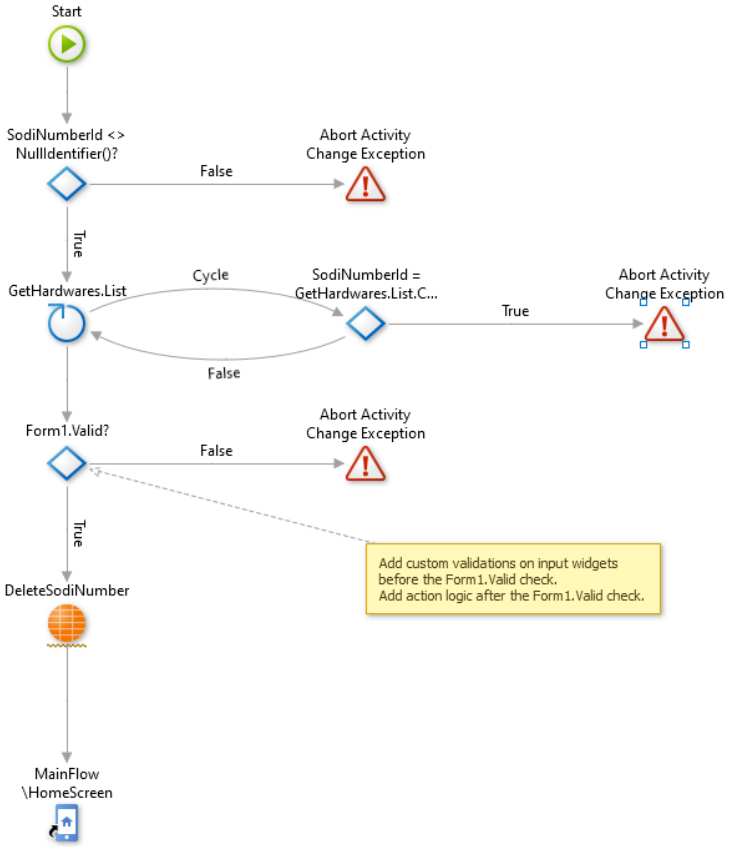
Jedná se o akce, které jsou specifické pouze pro obrazovky AddHardware a AddSodi. Je zde naimplementována logika skrývající se pod tlačítky Smazat, která maže data v databázi. Jako příklad bude popsána akce Smazat na obrazovce AddSodi.

Na obrázku 5.15 je možné vidět použití akce DeleteSodiNumber. Tato akce odebere existující záznam entity z databáze. Aby odebrání dopadlo v pořádku je potřeba nastavit podmínky odebrání a případně vytvořit chybové hlášky. První podmínka kontroluje, zda vstupní parametr SodiNumberId má již vytvořené ID. Pokud ne, tak aplikace vypíše hlášku „Nelze smazat neexistující protokol“. Jestli ano, tak se postupuje dále do cyklu For Each s podmínkou, která kontroluje, zda v entitě Hardware existuje záznam se stejnou hodnotou atributu SodiNumberId jako je hodnota vstupního parametru. Tím je eliminována možnost smazání protokolu, který obsahuje záznamy sériových čísel. Aby mohl být protokol smazán, musí být nejdříve smazány všechny záznamy sériových čísel obsahující

ID protokolu. Pokud podmínka skončí s hodnotou True, činnosti se ukončí a aplikace vypíše chybovou hlášku „Nelze smazat protokol, který v sobě obsahuje záznamy“. V opačném případě aplikace zkontroluje validaci vstupů a smaže záznam. Akce je ukončená přesunem na hlavní obrazovku.



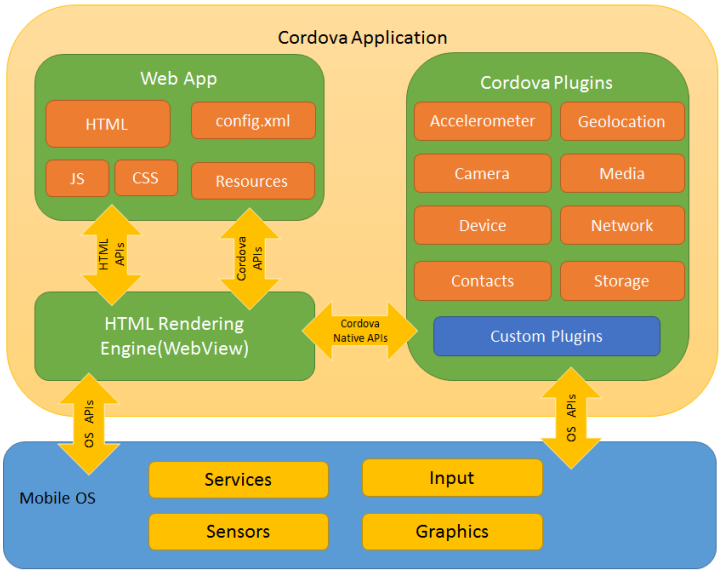
Obrázek 5.14: Akce Uložit



Obrázek 5.15: Akce Smazat

5.6 Implementace čtečky

Pro implementaci čtečky do aplikace bylo potřeba stáhnout Barcode Plugin z OutSystems Forge. OutSystems nabízí přes 3000 pluginů, které jsou vytvořené jak samotnými vývojáři platformy, tak i komunitou. Pluginy jsou založené na technologii Apache Cordova. Apache Cordova je open-source mobilní vývojový framework, který umožňuje vytvářet mobilní aplikace pro platformy iOS a Android pomocí standartních webový technologií – HTML5, CSS3, JavaScript. Takto vytvořená aplikace se následně „zabalí“ a chová se jako nativní aplikace pro příslušnou platformu. Na obrázku 5.16 je znázorněna architektura technologie Apache Cordova. Cordova dokáže skrze pluginy zpřístupnit systémové API cílového zařízení pro využití fotoaparátu, GPS atd. Zmíněné vlastnosti využívá platforma OutSystems při tvorbě pluginů. Platforma dovoluje upravovat jak komunitní pluginy, tak vytvářet své vlastní pluginy. Možnost tvorby pluginů však vyzkoušena nebyla (Foundation, 2015).



Obrázek 5.16: Apache Cordova – Architektura frameworku (Foundation, 2015)

Barcode Plugin je oficiální plugin vytvořený vývojáři platformy OutSystems. Podporuje mnoho populárních symbolů (typů čárových kódů) včetně EAN-13/UPC-A, UPC-E, EAN-8, kódu 128, kódu 39 a QR kódu. Jedna z nutných podmínek instalace čtečky, je stažení Common Plugin od totožných tvůrců. Pro použití čtečky je potřeba oba pluginy připojit k aplikaci. Pluginy se připojují k hlavnímu modulu aplikace stejným způsobem jako datový modul (obrázek 5.4). Po připojení jsou vytvořeny v hlavním modulu 2 klientské akce – CheckBarcodePlugin a ScanBarcode.

CheckBarcodePlugin kontroluje, zda je správně načtený Cordova plugin a jestli má přístup k fotoaparátu zařízení. Akce obsahuje dva výstupní parametry. Parametr IsAvailable má datový typ Boolean. V případě přístupného pluginu vrací hodnotu True. Parametr Error má datový typ Text. Obsahuje detailní informaci o chybě.

ScanBarcode spouští čtečku čárových kódu. Obsahuje 3 vstupní a 3 výstupní parametry. Pomocí vstupních parametrů lze nadefinovat nastavení čtečky. Tím je myšleno, zda se bude otevírat přední nebo zadní kamera, zda se zapne svítilna nebo vypíše text zobrazující se na vybrané kameře. Výstupní parametr ScanResult obsahuje výsledek skenu a má datový typ Text. Parametr Success má datový typ Boolean. Ten vrací hodnotu True, pokud byl výsledek skenu byl úspěšný. Parametr Error má datový typ Text. Obsahuje detailní informaci o chybě.

Aby bylo možné čtečku využívat, je potřeba vytvořit tlačítka, pod kterými můžeme implementovat logiku. Jako příklad je vybrána implementace tlačítka „Scan“ z obrazovky AddHardware (obrázek 5.11). Tlačítka byla umístěna hned pod vstup příslušných atributů. Akce začíná zavoláním CheckBarcodePlugin. V podmínce IF je využit výstupní parametr IsAvailable:

```
CheckBarcodePlugin.IsAvailable
```

Pokud parametr skončí s hodnotou False, aplikace vypíše hlášku „Plugin není k dispozici“. Jinak se postupuje na další akci ScanBarcode. Vstupní parametry byly nastaveny na používání zadní kamery zařízení bez svítilny. Následně je v další podmínce využit výstupní parametr Success:

```
ScanBarcode.Success
```

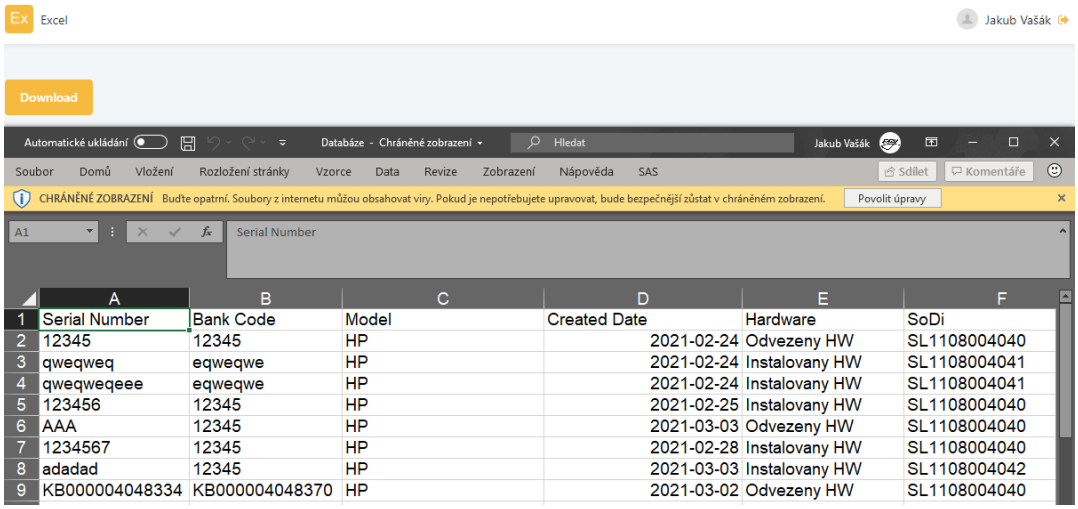
Jestli skončí podmínka s hodnotou False, potom se vypíše chybová hláška uložená v parametru Error. V opačném případě je pomocí funkce Assign příslušnému atributu přiřazená hodnota z výstupu čtečky a je vypsána hláška „Sériové číslo načteno“.

Obrázek 5.17: Implementace čtečky

5.7 Stažení databáze

Všechny vytvořené záznamy se ukládají do cloudové databáze, která je následně přístupná pouze administrátorovi. Databázi je potřeba stáhnout ve formě excelovského souboru. Pro získání databáze byla vytvořena webová aplikace sloužící pouze ke konvertování databáze do excelovského souboru a jeho následného stažení. K modulu webové aplikace byl připojen modul SnTCore, který obsahuje veškerá potřebná data. Aplikace obsahuje jednu serverovou akci a jednu obrazovou akci.

Serverová akce ConvertToExcelList vytvoří agregát GetHardwares obsahující entity Hardware, SodiNumber a Category. Pomocí funkce RecorListToExcel vybereme požadované atributy, které se budou konvertovat do excelovského souboru. Soubor je následně uložen ve výstupním parametru ExcelSheet. Na hlavní obrazovce bylo vytvořeno tlačítko „Download“ skrývající obrazovou akci, která zavolá serverovou akci ConvertToExcelList a umožní stáhnout do počítače výstup z parametru ExcelSheet pod jménem Databáze.csv. Na obrázku 5.18 lze vidět podobu výsledné webové aplikace a excelovského souboru se všemi atributy.



	A	B	C	D	E	F
1	Serial Number	Bank Code	Model	Created Date	Hardware	SoDi
2	12345	12345	HP	2021-02-24	Odvezeny HW	SL1108004040
3	qweqweq	eqweqwe	HP	2021-02-24	Instalovany HW	SL1108004041
4	qweqweqeee	eqweqwe	HP	2021-02-24	Instalovany HW	SL1108004041
5	123456	12345	HP	2021-02-25	Instalovany HW	SL1108004040
6	AAA	12345	HP	2021-03-03	Odvezeny HW	SL1108004040
7	1234567	12345	HP	2021-02-28	Instalovany HW	SL1108004040
8	adadad	12345	HP	2021-03-03	Instalovany HW	SL1108004042
9	KB000004048334	KB000004048370	HP	2021-03-02	Odvezeny HW	SL1108004040

Obrázek 5.18: Výsledná databáze v Excelu

5.8 Souhrn

V této kapitole byla popsána funkčnost a tvorba aplikace SnT Scanner. Aplikace byla vytvořena ve stanovaném čase 35 hodin i se studiem materiálů. Z 35 hodin bylo přibližně 8 hodin stráveno studiem. Dle mého názoru je nejdůležitější se naučit základy ve vývojovém prostředí OutSystems Studio 11, což vyžaduje čas. Je zde zabudováno mnoho možností a funkcí, díky nimž je zvykání a osvojování vývojového prostředí časově náročnější. Jde opravdu jen o to zvyknout si na prostředí, v čemž velice pomáhají kurzy a nápovědy. Po zvládnutí základů je přidávání dalších funkcionalit značně jednodušší. Vývojové prostředí nabízí funkci tzv. „1-Click Publish“. Tato funkce obsahuje fáze Ověření, Uložení na Cloud a Publikování aplikace. Pokud se během ověřování vyskytnou nějaké chyby, proces se zastaví a vyskočí okno, které ukáže, jaké chyby je pro pokračování potřeba opravit. Po úspěšném ověření se aplikace uloží a publikuje na cloud, kde je možné ji vyzkoušet. Cloud je dostupný na portálu OutSystems LifeTime, kde byla provedena i většina testování aplikace. Po implementaci čtečky byla vytvořena nativní aplikace pro platformu Android, kde bylo následně provedeno finální testování na poskytnutých čárových kódech. Při testování bylo objeveno několik chyb. Chyby se hlavně týkaly případů, kdy se ukládaly nebo mazaly záznamy z databáze. Všechny nalezené nedostatky byly opraveny podmínkami, jenž byly rozebrány v této kapitole. OutSystems Studio 11 nabízí při práci s kódem CSS a JavaScript zvýraznění syntaxe (což znamená barevné schéma v editoru kódu). Vytvořená aplikace nepotřebovala žádné výrazné zásahy do kódu. Nutno podotknout, že při vytváření složité aplikace musí vývojář disponovat znalostmi JavaScriptu. S tím souvisí i další nevýhoda užívání low-code platformy. Pokud vývojář programuje pomocí Pythonu, C# nebo jiného programovacího jazyka a narazí na problém, většinou si stačí vyhledat problém na Googlu. Často lze najít mnoho lidí, kteří již dříve zaznamenali stejný problém a nalezli na něj odpověď. Toto zde bohužel neplatí, protože low-code platformy stále nevyužívá tolik lidí jako standartní programovací jazyky. Pokud tedy vývojář narazí na chybu, kterou nedokáže vyřešit, a odpověď nenajde na fóru platformy OutSystems, musí vytvořit nové vlákno a počkat do té doby, než některý ze zaměstnanců platformy odpoví.

Celková zkušenost vývoje aplikace pomocí platformy OutSystems byla velice pozitivní. Po těžších začátcích bylo opravdu vidět, jak rychlý dokáže být vývoj pomocí low-code platformy. Zároveň celková stabilita platformy a nabízeného cloudu byla

perfektní. Vývojáři navíc neustále přidávají nejrůznější vylepšení skrze pravidelné aktualizace.

Závěr

V první části bakalářské byla stručně probrána historie programování, programovacích jazyků a následně vysvětlen pojem low-code. Byla zde představena hlavní myšlenka skrývající se za vývojem low-code společně s krátkou historií. Na základě tržní analýzy a osobních zkušeností byly vybrány dvě platformy, OutSystems a Mendix, které byly následně popsány v další kapitole. Zmíněné platformy se považují za leadery na low-code trhu díky skvělému zpracování a dobrým prodejům. V další části byl vysvětlen pojem low-code platforma a popsány metody a technologie, které jsou zde využity. Následně byly prozkoumány ukázkové vývojové platformy včetně jejich výhod a nevýhod. Na základě poznatků byla vybrána platforma OutSystems jako vývojové prostředí pro tvorbu zkouškové aplikace. Platforma OutSystems je jedna z nejstarších a nejvyspělejších vývojových platforem, která nabízí zdarma vývojovou edici, rozsáhlé návody, kurzy, dokumentace a portál pro stažení pluginů. Dále bylo srovnáno programování v low-codu s tradičním vývojem aplikací. Na příkladu byla rozebrána možnost ušetření finančních a časových prostředků při investice do jedné z těchto platforem. Poslední část se věnovala představením základních požadavků na vyvíjenou aplikaci společně s popsáním implementace ukázkové aplikace, určenou pro firmu S&T, pouze pomocí grafického vývoje při psaní žádného nebo pouze minimálního kódu. Veškeré firmou stanovené požadavky byly splněny. Finální aplikace umí číst čárové kódy jednotlivých instalovaných zařízení a ukládá je do databáze. Výsledná databáze může být exportována do souboru CSV.

Low-code platformy využívají podobné koncepty jako tradiční programování. Obě představené platformy mají dobře zpracované učební cesty a dokumentace, které zrychlují seznámení s využíváním platforem. Průměrná doba školení v platformě OutSystems je mnohem kratší než doba školení pro jakýkoliv jiný tradiční programovací jazyk. Jako příklad může být uveden jazyk C++ nebo C#, kde programátoři musí studovat a sbírat zkušenosti i několik let, aby z nich vznikli dobří vývojáři. Low-code je jednodušší a rychlejší přístup k vývoji aplikací. Ačkoliv je nutné podotknout, že nejlepších výsledků lze dosáhnout za podmínky, že osoba proškolená v low-code platformě má alespoň nějaké IT základy. Na rozdíl od náročných požadavků na kódování u tradičního vývoje low-code využívá rozhraní drag-and-drop, což umožňuje rychlejší vývoj a dodání. Low-code navíc nabízí mnoho funkcí a výhod pro podniky, které hledají efektivní nástroj pro vývoj aplikací. Podniky jsou hlavní cílovou skupinou low-code platforem. Tradiční vývoj vyžaduje

kvalifikované týmy a hluboké znalosti technologií a frameworků. Low-code nabízí snadnější vytváření aplikací a umožňuje i vývojářům s různou úrovní zkušeností dodávat aplikace v požadované časové ose. Tím se snižuje vytíženost IT specialistů a umožňuje jim věnovat svůj čas složitějším úkolům. Je zde navíc kladen větší důraz na zpětnou vazbu od uživatelů než na dodržování přísného plánu. Low-code upřednostňuje rychlé prototypování nad nákladným plánováním.

Low-code je ideální pro případy obchodního použití. Ve většině podniků chtějí uživatelé z oddělení jako HR, marketing atd. implementovat nápady na zlepšení fungování svého oddělení. Tradiční vývoj nemusí takovému scénáři vyhovovat, protože je méně přizpůsobivý neustálým změnám. V takových případech může být low-code skvělou alternativou k vývoji aplikací. Umožňuje rychle provádět změny a pokud tyto změny nefungují, lze je stejně snadno vrátit zpět. To dává možnost podnikovým uživatelům z různých oddělení volně provádět změny v aplikacích, aby vyhovovaly jejich aktuálním potřebám. Další případem využití může být rychlý vývoj aplikací pro podnikové uživatele. IT oddělení má obvykle seznam priorit a nemůže vytvářet speciální aplikace pro podnikové uživatele. Low-code platformy umožňují podnikovým uživatelům s malými nebo žádnými schopnostmi programování navrhovat aplikace podle jejich požadavků. Jako příklad může být uvedena právě vytvořená aplikace na inventarizaci zařízení. Pomocí low-codu lze také vytvářet aplikace, které se mění postupným vyvíjením požadavků koncového klienta firmy. Rozsah aplikace lze rozšířit tak, aby odrážel rostoucí potřeby klientů a zajišťoval optimalizaci nákladů. Navíc vývojové platformy můžou být i velice užitečné při vytváření prostředků pro správu a zpracování velkého množství dat. V situacích, kdy běžně dostupné programy nefungují a tradiční vývoj by byl až moc nákladný, se může low-code ukázat jako levnější, efektivnější a přizpůsobitelnější řešení.

Tradiční vývoj však má stále své místo. Low-code nejlépe funguje pro řešení, která sledují definovanou strukturu a musí být odpovídajícím způsobem navržena. Pro problémy, které jsou otevřenější a nestrukturované, musí být využit tradiční vývoj. Dalším příkladem využití tradičních programovacích jazyků je u aplikací, které potřebují velmi upravené funkcionality, kde je prioritou vysoký stupeň zabezpečení s integrací s několika systémy. Ačkoli low-code platformy mohou vytvářet vysoce efektivní aplikace, neposkytují flexibilitu a svobodu, kterou poskytuje tradiční vývoj při vytváření aplikací. Důvodem je, že tradiční vývojové platformy poskytují nekonečné množství funkcí.

Nelze tedy jednoznačně říci, zda vývoj pomocí low-codu je lepší než tradiční vývoj. Pravdou je, že low-code platformy přijaly osvědčené vývojové metody, které vylepšily. Společně s inovativními funkcemi vytvořili nový, rychlejší způsob poskytování aplikací. Tradiční programování však stále nabízí bezkonkurenční svobodu a flexibilitu. Podniky mohou těžit z programů vytvořených na míru potřebám firmy, a to někdy je přesně to, co daná společnost potřebuje. Proto by se před investicí do low-code platformy mělo pečlivě zvážit, zda je aplikace vhodná pro vývoj pomocí low-codu.

Seznam použité literatury

Almásy, Peter. 2001. *Haskell a funkcionální programování.* [Online] root.cz, 26. Září 2001. [Citace: 20. Prosinec 2020.] https://www.root.cz/clanky/haskell-a-funkcionalni-programovani/.

Čada, Ondřej. 2009. *Objektové programování.* Praha : © Grada Publishing, a.s. 2011, 2009. ISBN 978-80-247-6699-7.

Černohorský, Pavel. 2003. *Historie a vývoj jazyka C (od C až po C#).* [Online] Masarikova univerzita, Fakulta informatiky, 2003. [Citace: 25. Listopad 2020.] https://www.fi.muni.cz/usr/jkucera/pv109/2003p/xcernoh1.htm.

Dearmer, Abe. 2021. *What is Low-Code? Low-Code vs. No-Code, Low-Code Development Tools, and More.* [Online] xplenty.com, 2021. [Citace: 27. Leden 2021.] https://www.xplenty.com/blog/what-is-low-code/#developed.

Forsyth, Alexander. 2021. *What Is Low-Code?* [Online] www.outsystems.com, 2021. [Citace: 25. Leden 2021.] https://www.outsystems.com/blog/posts/what-is-low-code/?sc_lang=en.

Foundation, The Apache Software. 2015. *cordova.apache.org.* [Online] The Apache Software Foundation, 2015. [Citace: 25. Únor 2021.] https://cordova.apache.org/docs/en/latest/guide/overview/#architecture.

Hájek, Jaroslav. 2007. *Fortran: chroustání čísel pro všechny.* [Online] root.cz, 30. Červenec 2007. [Citace: 2020. Prosinec 15.] https://www.root.cz/clanky/fortran-chroustani-cisel-pro-vsechny/.

Kocan, Marek. 1998. *Víte, co je SQL? Ne? Nevadí - dnes začínáme!* [Online] zive.cz, 26. Říjen 1998. [Citace: 15. Prosinec 2020.] https://www.zive.cz/clanky/vite-co-je-sql-ne-nevadi---dnes-zaciname/sc-3-a-4320/default.aspx.

Kříž, Pavel. 2002. *Vývoj programování a programovacích jazyků.* [Online] Masarikova univerzita, Fakulta informatiky, 2002. [Citace: 20. Listopad 2020.] https://www.fi.muni.cz/usr/jkucera/pv109/2002/xkriz1.htm.

Kučera, Jan. 2000. *Historie programovacích jazyků.* [Online] Masarikova univerzita, Fakulta informatiky, 2000. [Citace: 20. Listopad 2020.] https://www.fi.muni.cz/usr/jkucera/pv109/2000/xkrubova.htm.

Mendix. *mendix.com.* [Online] © Mendix Tech BV. [Citace: 25. Leden 2021.] www.mendix.com.

—. *mendix.com.* [Online] © Mendix Tech BV. [Citace: 25. Leden 2020.] https://www.mendix.com/evaluation-guide/enterprise-capabilities/runtime-architecture/.

Mooney, Ann. *How Much Does Custom Software Cost?* [Online] SOLTECH. [Citace: 30. Leden 2021.] https://soltech.net/how-much-does-custom-software-cost/.

OutSystems. *outsystems.com.* [Online] OutSystems ©. [Citace: 25. Leden 2021.] https://www.outsystems.com/evaluation-guide/platform-runtime/.

—. *outsystems.com.* [Online] OutSystems ©. [Citace: 25. Leden 2021.] https://www.outsystems.com/.

Parker, Even. 2020. *Low-Code vs. No-Code: The Real Difference.* [Online] xplenty.com, 5. Únor 2020. [Citace: 25. Leden 2021.] https://www.xplenty.com/blog/low-code-vs-no-code/.

Pecinovský, Rudolf. 2009. *Objektově orientované programování? Co to je?* [Online] http://vyuka.pecinovsky.cz/, 2009. [Citace: 25. Listopad 2020.] http://vyuka.pecinovsky.cz/prispevky/2009_PS_OOP_-_Co_to_je.pdf.

Pekař, Lukáš. 2019. *FUNKCIONÁLNÍ PROGRAMOVÁNÍ.* [Online] https://bonsai-development.cz/, 2019. [Citace: 15. Prosinec 2020.] https://bonsai-development.cz/clanek/funkcionalni-programovani.

Polanský, Dušan. 2012. *Strukturované programování.* [Online] dusanpolansky.cz, 2012. [Citace: 22. Listopad 2020.] http://www.dusanpolansky.cz/clanky/banker.html.

Richardson, Clay a Rymer, John. 2014. *New Development Platforms Emerge For Customer-Facing Applications.* [Online] forrester.com, 9. Červen 2014. [Citace: 25. Leden 2021.]
https://www.forrester.com/report/New+Development+Platforms+Emerge+For+CustomerF

acing+Applications/-/E-RES113411?utm_source=xp&utm_medium=blog&utm_campaign=content.

Ross, Malcom. 2018. *4 essential features of modern low-code development platforms.* [Online] https://www.infoworld.com/, 2018. [Citace: 25. Leden 2021.] https://www.infoworld.com/article/3287146/4-essential-features-of-modern-low-code-development-platforms.html.

Stočes, Michal. 2019. *Programovací jazyky, Přehled a vývoj.* [Prezentace] Praha, Česká zemědělská univerzita : Česká zemědělská univerzita, 2019.

Škvarda, Libor. 2006. *Co je to funkcionální programování.* [Online] http://programujte.com/, 2006. [Citace: 10. Prosinec 2020.] http://programujte.com/clanek/2006032503-co-je-to-funkcionalni-programovani/.

Tišnovský, Pavel. 2020. *Šedesátiny převratného programovacího jazyka ALGOL-60.* [Online] root.cz, 2020. [Citace: 20. Listopadu 2020.] https://www.root.cz/clanky/sedesatiny-prevratneho-programovaciho-jazyka-algol-60/.

—. 2009. *Programování mainframů: COBOL.* [Online] root.cz, 2009. [Citace: 2020. Listopadu 20.] https://www.root.cz/clanky/programovani-mainframu-cobol/.

—. 2019. *Jemný úvod do rozsáhlého světa jazyků LISP a Scheme.* [Online] https://www.root.cz/, 2019. [Citace: 15. Prosinec 2020.] https://www.root.cz/clanky/jemny-uvod-do-rozsahleho-sveta-jazyku-lisp-a-scheme/.

—. 2019. *Sedmdesátiny assemblerů: lidsky čitelný strojový kód.* [Online] root.cz, 1. Říjen 2019. [Citace: 25.. Prosinec 2020.] https://www.root.cz/clanky/sedmdesatiny-assembleru-lidsky-citelny-strojovy-kod/.

van Oosten, Arjo. 2020. *Achieve New Levels of Business Value with Low Code.* [Online] www.mendix.com, 11. Červen 2020. [Citace: 30. Leden 2021.] https://www.mendix.com/blog/achieve-new-levels-of-business-value-with-low-code/.

Vincent, Paul, a další. 2020. *Gartner Magic Quadrant for Enterprise Low-Code Application Platforms.* [Online] gartner.com, 30. Listopad 2020. [Citace: 25. Leden 2021.] https://www.gartner.com/doc/reprints?id=1-24BBDEDZ&ct=201005&st=sb.

Vostrovský, Václav. 2014. *Vytváření databází v Oracle.* Praha : Česká zemědělská univerzita v Praze, 2014. ISBN 978-80-213-1191-6.

Wilfrid, Davin. 2020. *A Brief History of Low-Code Development Platforms.* [Online] https://www.quickbase.com/, 2020. [Citace: 27. Leden 2021.] https://www.quickbase.com/blog/a-brief-history-of-low-code-development-platforms.

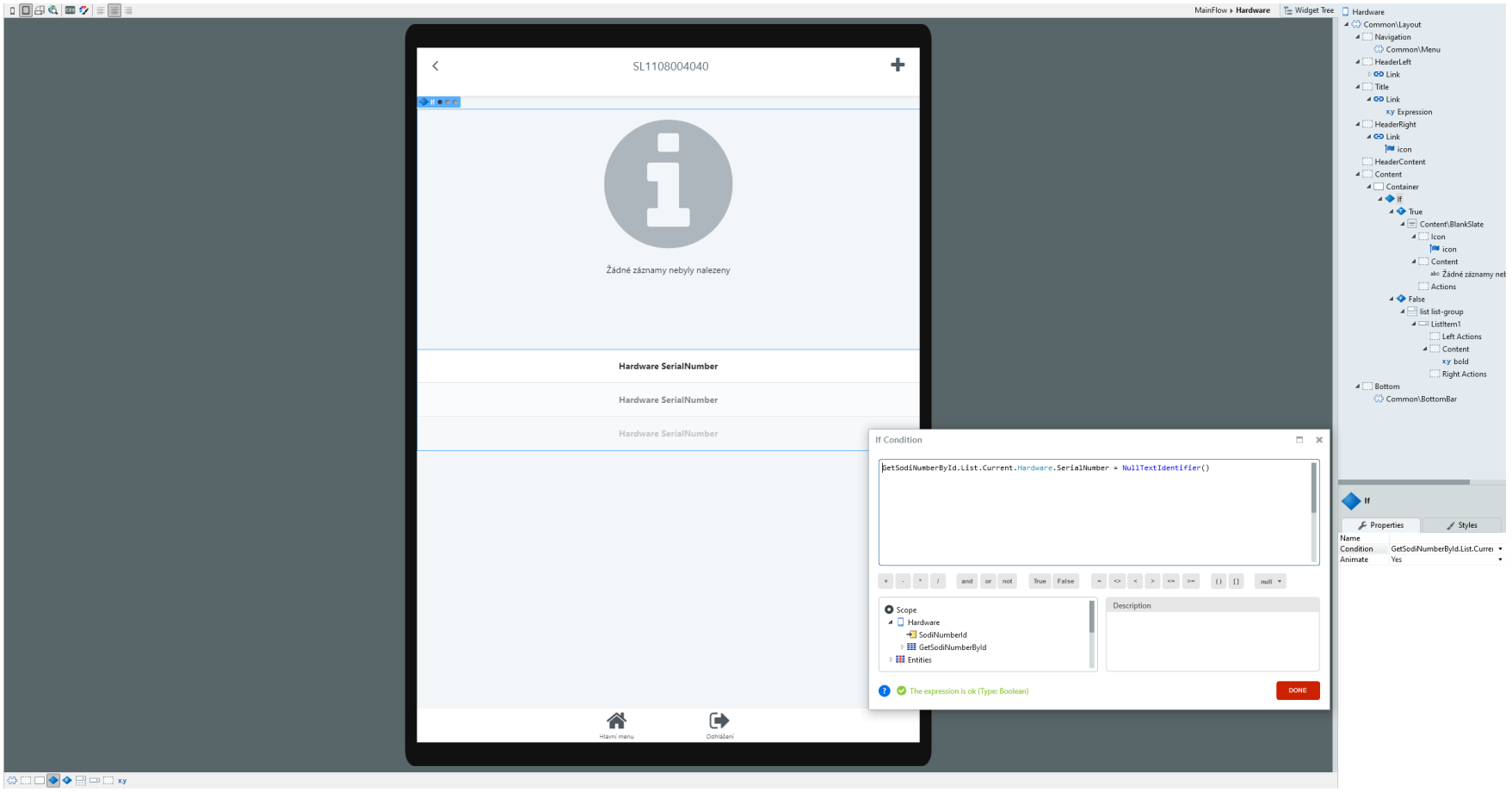
Žoltá, Lucie. 2013. *Deklarativní programování.* [Online] http://lucie.zolta.cz/, 2013. [Citace: 2020. Prosince 10.] http://lucie.zolta.cz/index.php/statnice-vsb.

Seznam obrázků

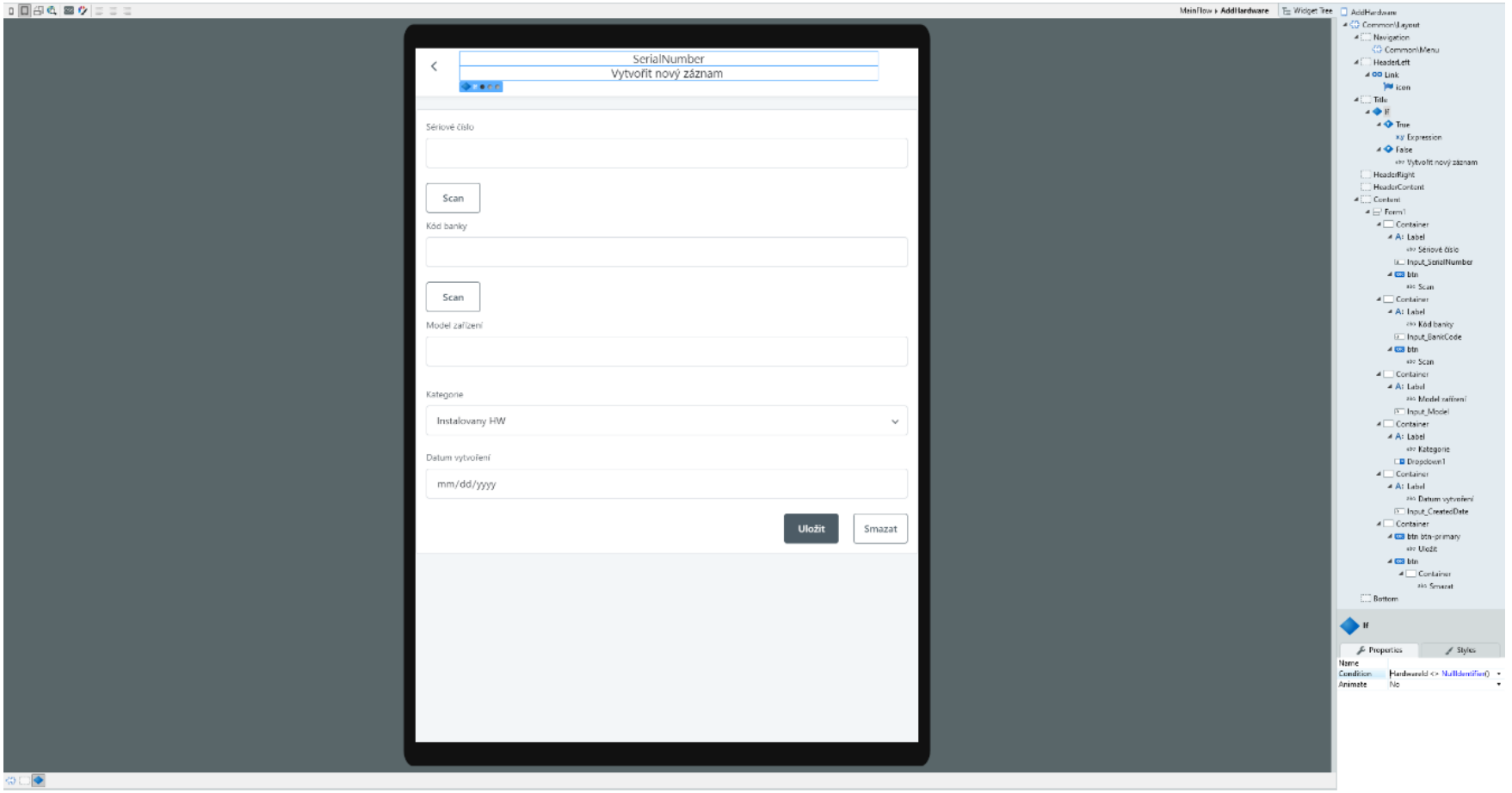
Seznam příloh

Přílohy

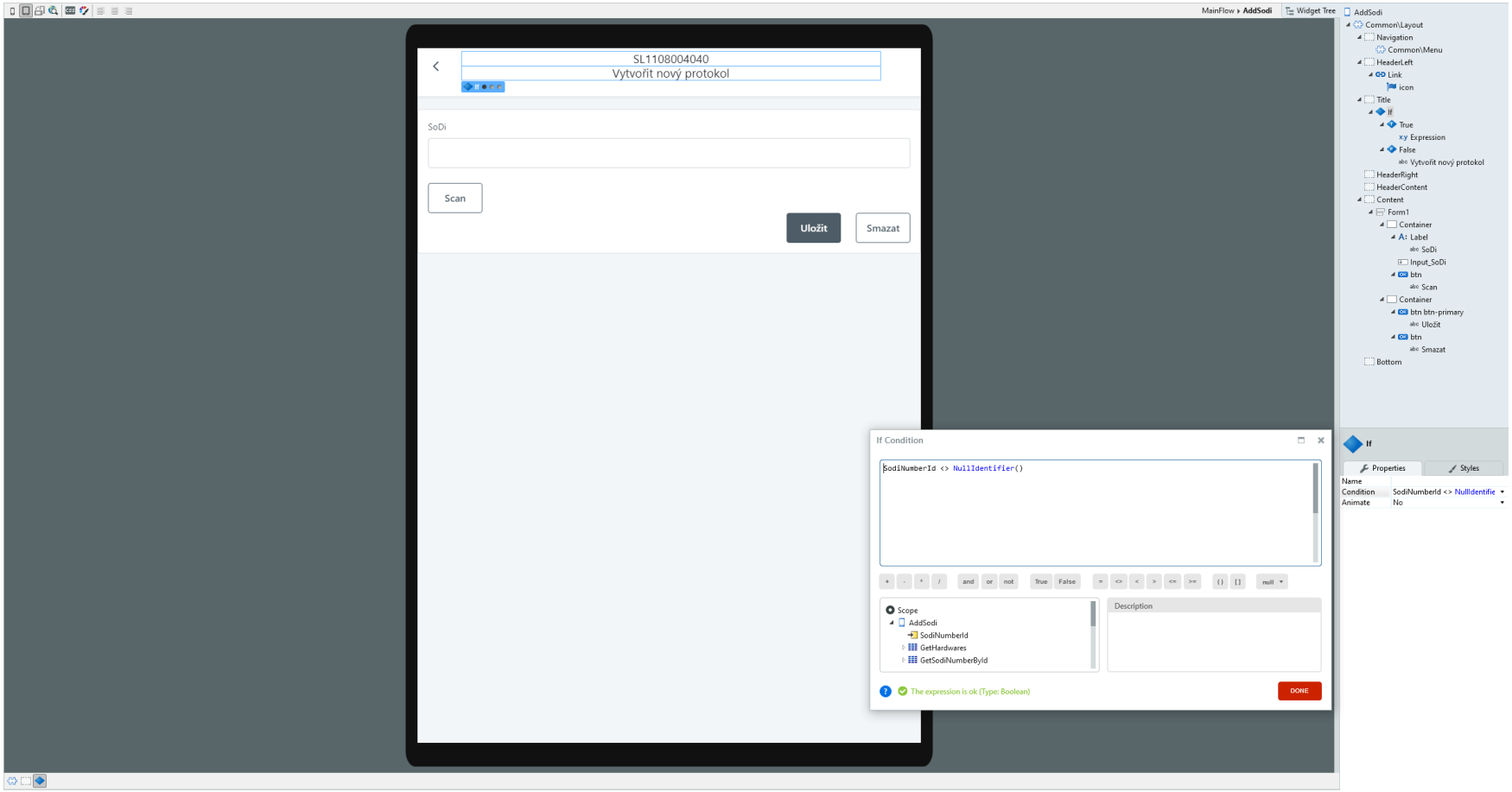
Příloha 1: Složení hlavní obrazovky



Příloha 2: Složení obrazovky Hardware

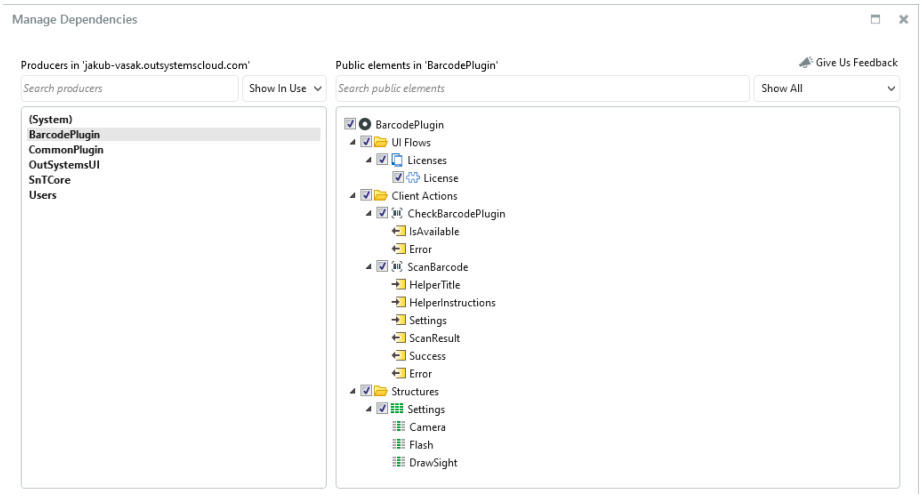


Příloha 3: Složení obrazovky AddHardware

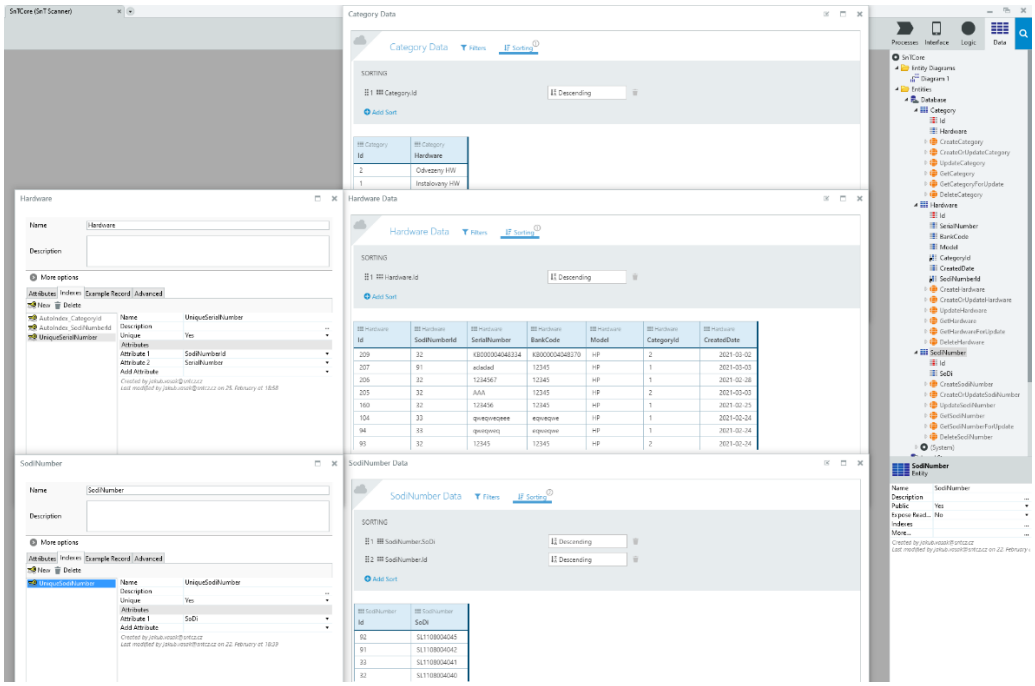


Příloha 4: Složení obrazovky AddSodi

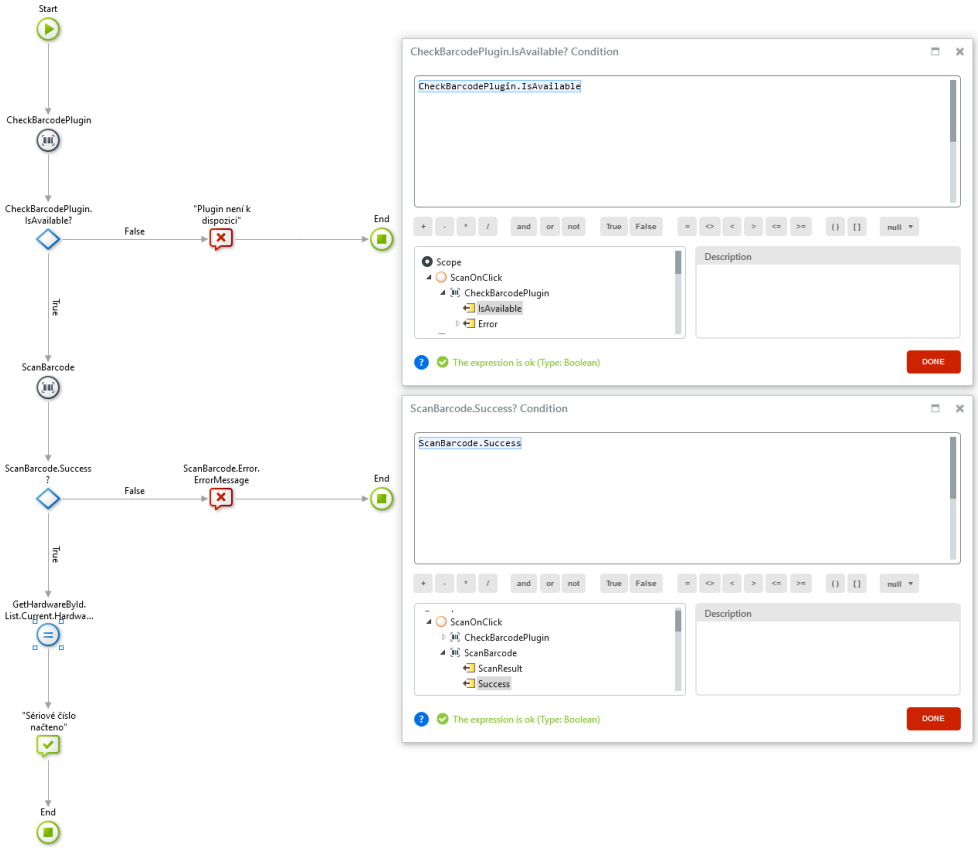
Příloha 5: Složení bloku BottomBar

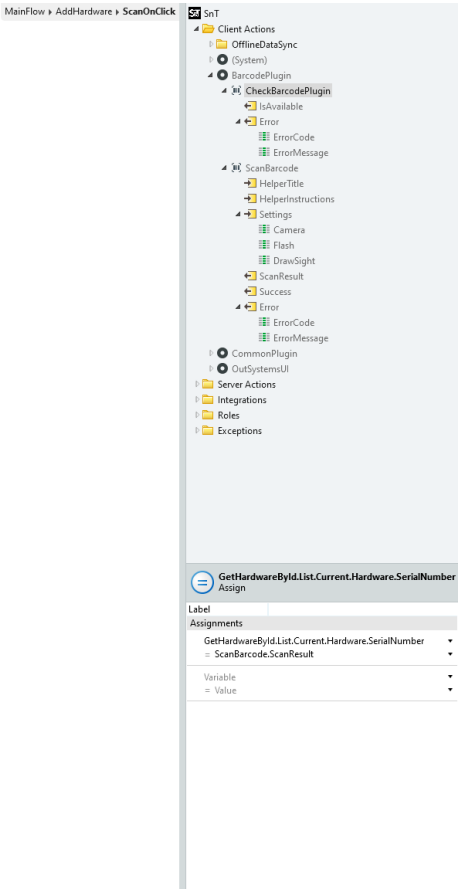


Příloha 6: Závislosti modulu SnT



Příloha 7: Složení entit

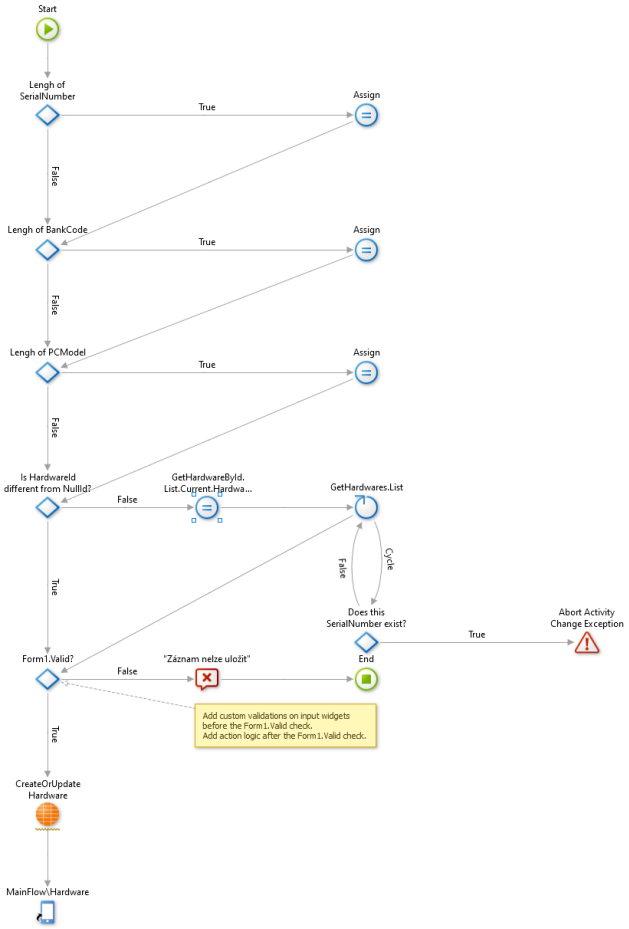


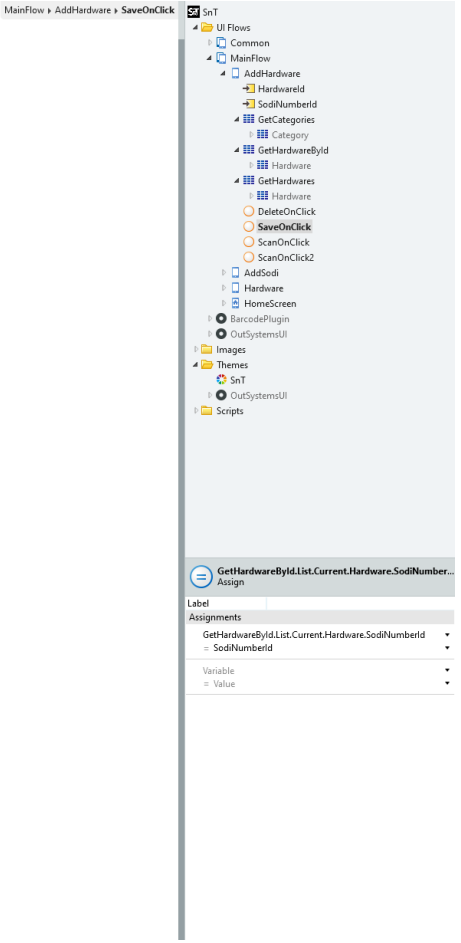


Příloha 8: Složení tlačítka Scan

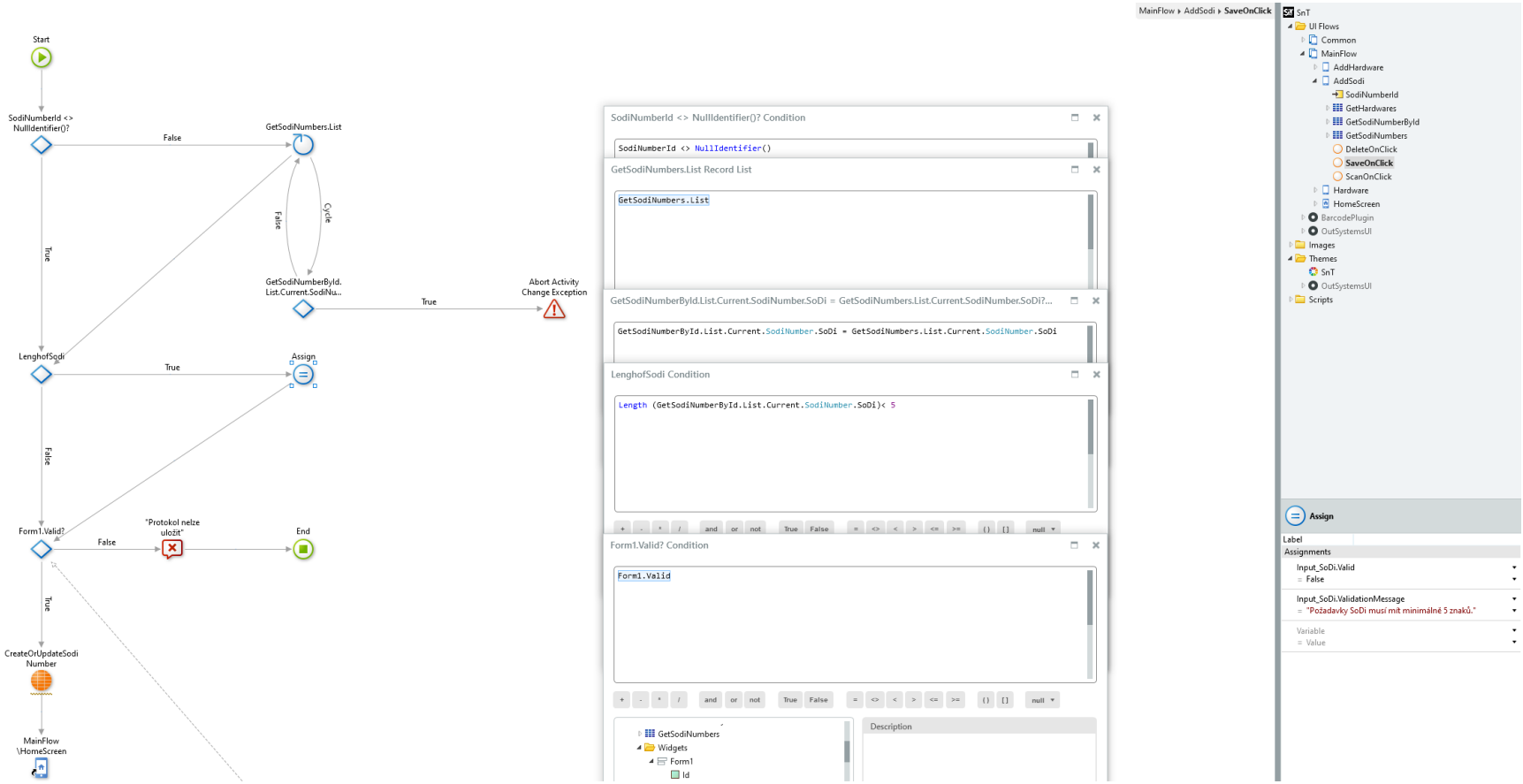
Příloha 9: Složení tlačítka Smazat – obrazovka AddHardware

Příloha 10: Složení tlačítka Smazat – obrazovka AddSodi



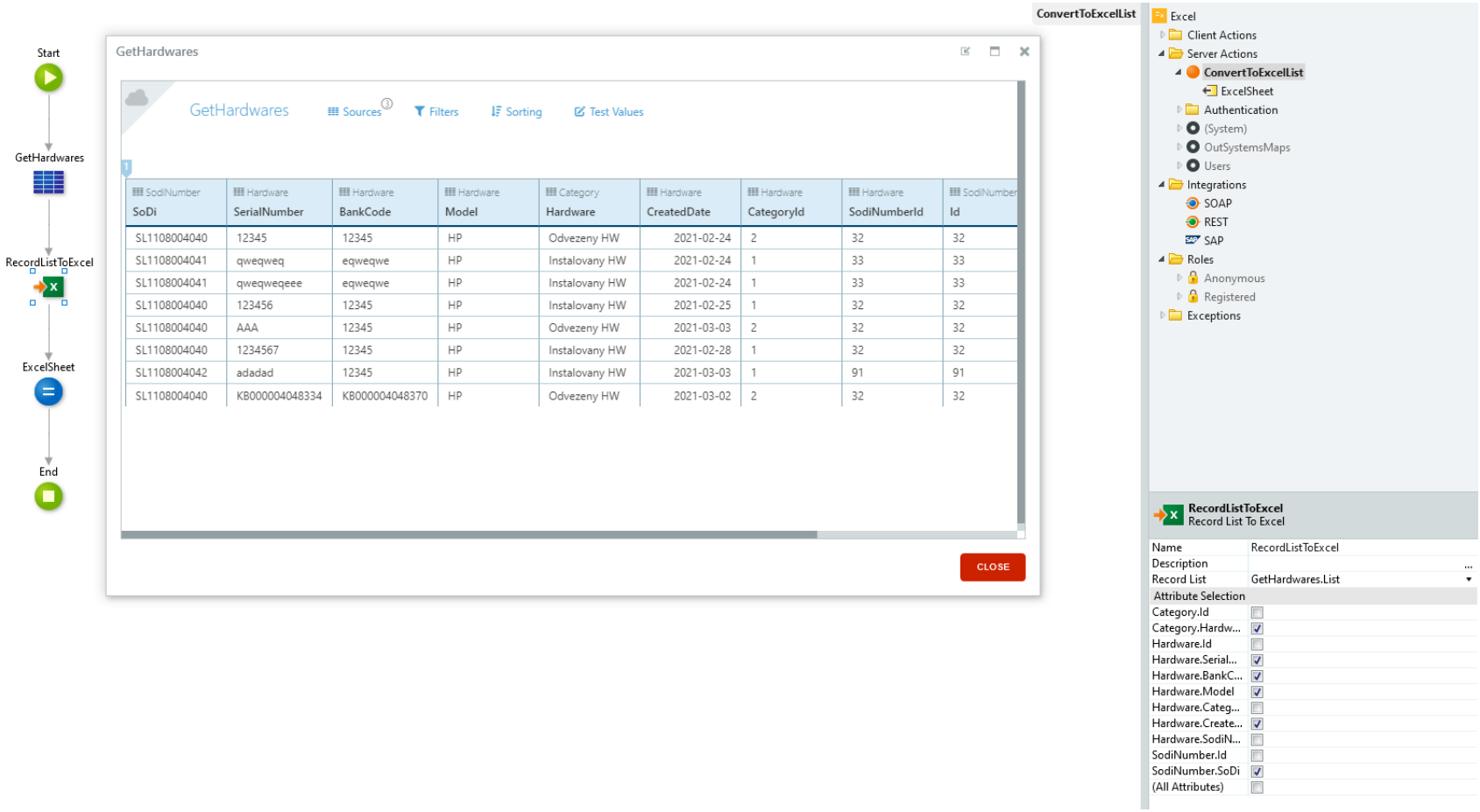


Příloha 11: Složení tlačítka Uložit – obrazovka AddHardware



Příloha 12: Složení tlačítka Uložit – obrazovka AddSodi

Příloha 13: Složení hlavní obrazovky aplikace pro stažení databáze



Příloha 14: Složení akce konvertující databázi do souboru CSV

Příloha 15: Složení tlačítka Download