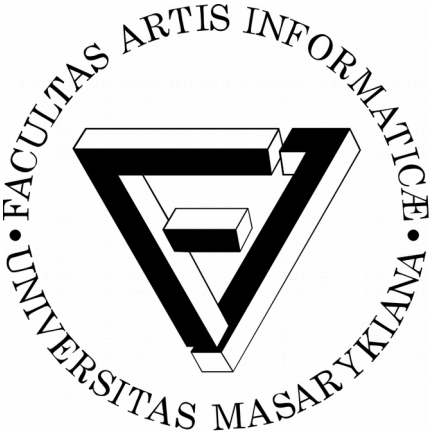
Masarykova univerzita

Fakulta informatiky



Tvorba výukových materiálov pre školenia frameworkov Angular 2 a Ionic

Bakalářská práce

Zuzana Wolfová

Brno, jaro 2018

Prohlášení o autorství

Prohlašuji, že tato práce je mým původním autorským dílem, které jsem vypracoval(a) samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal(a) nebo z nich čerpal(a), v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Shrnutí

Bakalářská práce se zabývá frameworkem Ionic, souvisejícími technologiemi Angular a Cordova, popisuje základní principy těchto technologií nutné pro zaučení juniorních programátorů. Práce se zároveň zabývá metodou výuky a školení v komerčním prostředí této technologie se zaměřením na efektivitu vzdělávání. Práce následně shrnuje prakticky získáné informace ze školení této technologie a osvědčené postupy.

Klíčová slova

Angular, Ionic, Cordova, multiplatformní aplikace, mobilní aplikace, výukové materiály

Obsah

1 Úvod....1
2 Výuka a školení informačních technologií v komerčních společnostech....4
2.1 Teoretický výklad....4
2.2 Praktická demonstrace....5
2.3 Samostatná práce....8
2.4 Hodnocení....8
3 Klíčové prvky technologií....10
3.1 Apache Cordova....11
3.1.1 Přijetí platformy....13
3.1.2 Použití frameworku....14
3.1.3 Distribuce aplikace....14
3.2 SPA....15
3.3 Angular....16
3.3.1 Podporované programovací jazyky....17
3.3.2 Mechanismus dependency injection....18
3.3.3 Propojení vlastností a událostí....18
3.3.4 Obousměrné propojení....19
3.4 Ionic....19
3.4.1 GUI komponenty....20
3.4.1.1 Vzhled komponent....20
3.4.1.2 Responzivní design....21
3.4.2 Programové komponenty....22
3.4.3 Zásuvné moduly....23
3.4.4 Podpůrné nástroje....23
4 Praktický úvod do vývoje v Ionic frameworku....24
4.1 Příprava projektu a prostředí....24
4.1.1 Instalace prostředí....24
4.1.2 Založení nového projektu....25
4.2 Práce s projektem....25

4.2.1 Spuštění vývojového serveru....27
4.2.2 Spuštění aplikace na mobilním zařízení....27
4.2.2.1 iOS....27
4.2.2.2 Android....28
4.3 Základy vývoje....28
4.3.1 Stránky....28
4.3.2 Služby....29
5 Závěr....30
Reference....32
A Přílohy....34
A.1 Obsah elektronické přílohy....34

Seznam obrázků

Obrázek 1: Architektura aplikace v Angularu 2+. [cit. 2018-05-27]. Dostupný z: https://angular.io/generated/images/guide/architecture/overview2.png.....17

1 Úvod

Problematika IT vzdělávání je komplexní činností díky mnoha specifikům tohoto oboru, např.:

- velmi široké a rozsáhlé problematice s mnoha specializacemi;
- dynamickému rozvoji a krátké životnosti technologií;
- nutnosti kombinace mnoha různých jazyků, knihoven a frameworků při vývoji projektů.

Nejen proto je vzdělávání v IT zcela klíčovou záležitostí a musí být součástí profesního rozvoje každého programátora, pokud mají být jeho znalosti relevantní i v následujících letech. Díky tomu je možné pozorovat i u zaměstnavatelů velký důraz na profesní vzdělávání svých pracovníků – ať už juniorních, kteří teprve získávají nutné profesní znalosti, tak seniorních pracovníků, aby se udrželi na technické špičce a neztratili kontakt s moderními trendy a technologiemi.

V rámci své práce se zabývám jedním z velmi perspektivních IT oborů – vývojem mobilních aplikací. Studie ukazují, že mobilní aplikace představují velmi rychle rozvíjející se trh, který s ústupem klasických PC a desktopových aplikací představuje stále důležitější segment pro společnosti. O tom svědčí i počet dostupných aplikací, který pro nejpopulárnější platformy iOS a Android dosahuje několika milionů.

Jedním ze zásadních problémů, se kterým se společnosti při vývoji mobilních aplikací potýkají, je nutnost vývoje pro různé platformy. Vývoj v iOS probíhá tradičně v jazyku Objective-C či Swift, pro platformu Android v jazyku Java či Kotlin. Odlišné programovací jazyky ovšem zdaleka nejsou jediným rozdílem, zcela zásadní je kompletní odlišnost prostředí, jednotlivých API (Application programming interface) a knihoven. S trochou nadsázky můžeme říci, že vývoj mobilní aplikace pro platformu iOS má s vývojem na systému Android společné pouze jediné – jedná se o vývoj mobilní aplikace.

Z tohoto důvodu získávají na popularitě technologie, které umožňují multiplatformní vývoj – tedy práci v jednom jazyce, technologii a frameworku, jejímž výsledkem

je aplikace použitelná na obou majoritních mobilních operačních systémech. Proto jsem pro svou práci vybrala framework Ionic, který přesně to umožňuje a zároveň patří k nejpopulárnějším multiplatformním mobilním technologiím.

Ionic využívá hned několik moderních technologií, které umožňují velké zjednodušení vývoje. Aplikace jako taková je postavena na známých a běžně používaných technologiích jako je HTML5, CSS3 a TypeScript (nadmnožina jazyka ECMAScript 6, nepřesně nazývaného JavaScript, který nicméně tvoří základ vývoje webových aplikací). Další velmi pozitivní vlastností této technologie je, že řeší i velmi častý problém vývoje multiplatformních aplikací – každý systém je odlišný v koncepci uživatelského rozhraní, jeho vzhledu a UX (User Experience). Ionic standardně poskytuje CSS styly specifické pro danou platformu s přizpůsobeným vzhledem, díky tomu vypadají ovládací prvky přirozeně, ať už se jedná o systém iOS, či Android. Ionic zároveň staví na frameworku Android (vyvíjeného společností Google), který patří spolu s technologií React k nejpoužívanějším technologiím pro vývoj webových (nejen mobilních) aplikací. Ionic přidává k frameworku Android zejména technologie a ovládací prvky nutné pro snadný vývoj mobilních aplikací, nicméně základní koncepty zůstávají zachovány. Bohužel ač se jedná o velmi užitečný a běžně používaný framework, trpí poměrně typickým neduhem podobných, rychle rozvíjejících se technologií - absencí aktuální literatury a dokumentace, zejména v češtině.

Ve své práci se tedy zabývám popisem frameworku (cílem je i poskytnout podpůrný výukový materiál s popisem nejdůležitějších vlastností) a zhodnocení výzev při učení této technologie pro programátory. Cílem práce je zejména:

- popsat základní aspekty problematiky vývoje mobilních aplikací;
- popsat specifika IT vzdělávání v komerčních společnostech;
- vytvořit úvod do frameworku Ionic a souvisejících technologií (Angular, Cordova a další);
- poskytnout základní potřebné znalosti a materiály pro programátory začínající využívat tuto technologii.

V práci se zaměřím na obecný popis jednotlivých technologií, jejich vztahů a architektury.

Následně se zaměřuji na technický popis, konkrétní postupy, při tvorbě základních prvků mobilní aplikace (např. založení projektu, kompilace pro jednotlivé platformy, základní programátorské koncepty). Výsledky práce slouží jako podpůrný výukový materiál při seznamování se s technologií Ionic. Součástí této práce byla i zejména tvorba výukových materiálu pro interní školení.

V závěru práce následně shrnuji hlavní výhody technologie Ionic, hodnotím složitost vzdělávání v tomto frameworku (pro vývojáře bez předchozích zkušeností s vývojem mobilních aplikací), získané vědomosti programátorů a stanovuji doporučení pro další postupy vzdělávání.

2 Výuka a školení informačních technologií v komerčních společnostech

Výuka a školení informačních technologií v komerčních společnostech (zejména programování ve specifických technologiích) má mnoho specifik, které je nutné reflektovat ve výukovém modelu:

- specifické, individuální vstupní znalosti;
- omezená časová dotace školení;
- tlak na co nejrychlejší integraci do vývojového týmu / seznámení s technologií;
- časté změny technologií, jejich rozvoj a nutnost zaučit se v krátkém čase do nových;
- snaha omezit opakování běžných chyb.

Díky tomu většina IT školení probíhá v tzv. režimu "hands on" (1), který se vyznačuje získáváním jak teoretických znalostí, tak praktických a nutí účastníky nově získané znalosti okamžitě prakticky aplikovat a používat.

Školení můžeme rozdělit do následujících bloků:

- teoretický výklad a vysvětlení problematiky, definice problému;
- praktická demonstrace s lektorem;
- samostatná práce;
- dotazy.

Cílem je tedy účastníky rychle seznámit s teoretickým základem problematiky, poskytnout jim základní zdroje pro další studium. Následně získané teoretické znalosti demonstruje lektor na praktickém příkladu (řešení předem stanoveného problému), poté lektor zadává samostatnou práci (varianta dříve řešeného problému) a sleduje postup účastníků.

2.1 Teoretický výklad

Teoretický výklad je důležitý a není možné jej přeskočit, nicméně je nezbytné respektovat specifika daného prostředí. Jedním ze zásadních problémů teoretického výkladu je udržení pozornosti účastníků a optimální balancování času mezi teoretický výklad

a praktické části školení (2).

Pro teoretický výklad bylo v rámci školení doporučeno:

- kratší výklad definující základní problémy a principy;
- vazba teoretických znalostí na praktické problémy, prokládání teoretického výkladu vazbami na praktické problémy;
- poskytování referencí na další zdroje, kde mohou účastníci získat podrobnější znalosti;
- pokud je to možné, používání vizuálních pomůcek, diagramů či grafů.

Cílem teoretického výkladu není poskytnout úplné znalosti problematiky na teoretické úrovni, ale zejména poskytnout nezbytné minimum pro správné pochopení probírané tématiky, jejího kontextu a vazeb a zároveň poskytnout referenci pro další vzdělávání v problematice, které již mohou zvládnout účastníci samostatně.

Samostatnou kapitolou je potom délka teoretického výkladu – z mých praktických zkušeností vyplynulo, že teoretický výklad delší než jedna hodina, je pro účastníky problematický, v tématu se začínají ztrácet a jejich pozornost klesá (3).

To představuje u mnoha témat zásadní problém – zejména tam, kde se účastníci seznamují s úplně novou technologií a nemohou navázat na své existující znalosti. Je proto nutné při návrhu školení přemýšlet nad vhodným rozdělením teoretického výkladu takovéto skupině, vybrat několik nejzásadnějších témat. To bohužel vede často k nepřesnostem a přílišným zjednodušením - na takové je ale vhodné účastníky upozornit, aby jasně pochopili, že to, co se nyní definuje, není komplexní pohled na situaci, ale zjednodušení, a že k dané problematice se skupina vrátí později v průběhu dalšího seznamování s technologií.

2.2 Praktická demonstrace

Teoretický výklad je nutný, nikoliv dostatečný --pro správné pochopení problematiky je nutné demonstrovat použití nově získaných teoretických vlastností. Vzhledem k

omezené časové dotaci nelze očekávat, že po teoretickém výkladu by byli schopni účastníci dané znalosti okamžitě samostatně použít a aplikovat – pravděpodobně byli, ale trvalo by to příliš dlouho, což při intenzivním vzdělávání není možné.

Je vhodné věnovat pozornost volbě praktické demonstrace, ta by měla v ideálním případě:

- přímo navazovat a rozvíjet teoretický výklad;
- vycházet z praktických problémů, se kterými se vývojáři budou setkávat.

Ze zkušeností je vhodné při praktické demonstraci opakovat některé znalosti a fakta z předchozího teoretického výkladu – student by nikdy neměl získat pocit, že teoretický výklad a praktická demonstrace jsou oddělenými (či v nejhorším případě nezávislými) částmi školení.

Vhodně zvolená praktická demonstrace tudíž nejen ukazuje praktické použití, ale přímo navazuje a dále rozvíjí teoretický výklad, dodává mu kontext a transformuje ho do problémů, které budou programátoři při své práci reálně muset řešit.

Důležitou součástí správné praktické demonstrace je i volba složitosti řešeného (demonstrovaného) problému a jeho řešení. Ze zkušeností je možné doporučit více různých praktických demonstrací, postupování od jednodušších (triviálních) problémů k těm složitějším. Účastník se v této části školení nesmí ztratit, musí pořád vědět jaký problém se řeší, jak ho řešíme a příp. i proč.

Při praktické demonstraci je třeba i věnovat pozornost interaktivitě (4) – účastník musí mít možnost se lektora dotázat na nejasnosti, lektor by naopak měl průběžně ověřovat, že účastníci chápou řešenou problematiku a proč zvolil takovéto řešení.

V případě, že existuje více ekvivalentních možností řešení, je na zvážení, zda na ně účastníky upozornit. Lze jednoznačně konstatovat, že poskytnout účastníkům hlubší kontext a různé alternativní možnosti řešení je pozitivní – upevňuje porozumění dané problematice.

Na druhou stranou je nutné mít určitý cit ze strany lektora – je zcela nevhodné, zvláště v případě obecnějších problémů a situací, zahltit účastníky myriádou různých možných řešení, byť se jedná o jakkoliv fakticky správnou informaci. V takových případech lektor musí vybrat nejdůležitější – ideální je v takovém případě:

- navazovat na existující, již probranou problematiku;
- vyhýbat se podobným/ekvivalentním řešením, volit spíše ty principiálně odlišné;
- vysvětlovat odlišnosti, výhody a nevýhody odlišných postupů, příp. (pokud je možné) vhodnost či nevhodnost v určitých specifických případech.

Častým problémem IT školení je i určitá disasociace mezi účastníky a probíranými tématy. Ne vždy si u parciálních kapitol každý účastník uvědomuje, k čemu může dané, nově nabité znalosti použít. Je tedy vhodné, zejména pokud má již lektor větší programátorské zkušenosti, prokládat praktický výklad určitými zmínkami z praxe – při jakých situacích tyto věci využil atd.

V této části se též jeví vhodné ověřovat určité již získané znalosti – v případě praktické demonstrace určitého problému zcela jistě lektor narazí na situaci, kdy je potřebné použít již něco z probraných technologií či jejich vlastností. V takovém případě se velmi hodí (ať již z důvodu procvičení či udržení pozornosti skupiny) neposkytnou celé řešení ihned, ale obrátit se na skupinu a nechat je zvolit pokračování řešení a nechat je problém vyřešit.

Pozornost je ovšem třeba věnovat heterogenním skupinám, v takovém případě může být nutné provést určité rozdělení na podskupiny podle znalostí. Tyto otázky by neměly být pro účastníky složité, ale měly by upevňovat jejich porozumění. Je normální, aby se nad nimi museli účastníci zamyslet, zanalyzovat možnosti a poté teprve odpověď. V heterogenní skupině toto může být problém z důvodu účastníků, kteří již mají větší vstupní vlastnosti. V takovém případě je potřeba tento proces ve skupině citlivě korigovat. Na jednu stranu je potřeba, aby na otázky odpovídali ti, kteří se tím ve vzdělávání posouvají, na druhou stranu není možné účastníky rozdělit na dva znepřátelené tábory, či úplně ztratit pozornost zdatnější části.

2.3 Samostatná práce

Samostatná práce účastníků je pedagogicky velmi důležitá – účastníci si zde ověří své parciální znalosti, lektor získá zpětnou vazbu o skupině, jak dané téma zvládli a pochopili (či nikoliv). Obecně je možné doporučit určitou korelaci mezi praktickou částí demonstrovanou lektorem a samostatnou prací.

Samostatná práce by obecně měla být jednodušší než problémy demonstrované lektorem, ale zároveň by měla vycházet z probírané látky. Zároveň by ovšem nemělo být možné takovou samostatnou práci splnit bez pochopení látky, je nutné tedy vhodně volit samostatné příklady tak, aby jejich obtížnost nebyla příliš složitá, ale ani triviální.

V průběhu práce je nutné dát možnost účastníkům se ptát a zodpovídat jejich dotazy, příp. jim pomoci s parciálním řešením. Cílem této části nemá být hodnocení, ale vzdělávání.

Poměrně často se lektor ovšem může setkat s účastníky, kteří jsou pasivní a i v případě, že neví jak pokračovat, problematice nerozumí, tak zůstávají pasivní a nekladou vhodné otázky. Proto je nutné s těmito účastníky pracovat a důsledně kontrolovat práci i výsledky všech, nikoliv jen těch aktivnějších účastníků.

Samozřejmě, v určitých případech ani tento přístup nepovede k úspěšnému zvládnutí problematiky – tyto případy je ovšem nutno řešit se společností a zajistit co nejrychlejší vyřazení těchto účastníků ze skupiny, jelikož se negativně projevují na vzdělávání celku (ať již díky času s nimi ztraceném či díky poklesu morálky a motivace ostatních).

2.4 Hodnocení

Hodnocení je velmi důležitou součástí každé vzdělávací aktivity. Nicméně stejně jako všechno – různé formy hodnocení mají svůj čas a své místo.

Při školení je nutné vytvořit přátelskou, pro-vzdělávací atmosféru, kde se účastníci nebojí klást otázky, což je velmi často problém z důvodu implicitního očekávání formální atmosféry, subjektivnímu strachu z kladení „hloupých" otázek. Avšak i v těchto počátečních fázích školení je hodnocení naprosto nezbytnou součástí – nicméně díky

negativnímu vnímání tohoto termínu je vhodné zvolit formulaci jinou – zpětná vazba. Cílem zpětné vazby je poskytnout náměty na zlepšení každého účastníka bez ohledu na to, zda plní interně funkci hodnocení či nikoliv.

Při hodnocení je nutné vyhnout se subjektivnímu hodnocení (implicitní bias), zejména v případě nestandardních řešení, každé řešení je nutné posoudit jednotlivě a neukvapovat se se závěry, hodnocení musí být komplexní se zvážením všech pro a proti.

Zároveň je vhodné ovšem definovat určité kontrolní body (1–2 např.), kdy studenti dostanou již větší prostor pro samostatnou práci, která plní hodnotící funkci a takto ji i vnímají studenti. Takováto práce musí reprezentativně umožnit zhodnotit komplexní pokrok v pochopení tématu, musí vyžadovat samostatnou práci i iniciativu. Student zároveň musí dostat vlastní prostor pro obhajobu svých rozhodnutí a možnost jejich vysvětlení.

Při hodnocení a poskytování zpětné vazby je velmi vhodný individuální přístup.

3 Klíčové prvky technologií

V této části jsou popsány klíčové technologie, které využívá framework Ionic a poskytuje základní teoretický rozhled pro programátora v tomto prostředí. Tato kapitola sloužila jako podpůrný výukový materiál a opora pro vzdělávání. Rámcově reflektuje i skutečnou návaznost témat při vzdělávání.

Framework Ionic patří mezi moderní technologie pro vývoj mobilních aplikací, které tvoří podstatnou část současného portfolia nabízených aplikací a představují pro vývojáře atraktivní rostoucí trh.

Jednou z velkých výhod frameworku je jeho multiplatformnost – aplikace je naprogramována jednou a je schopna provozu na mnoha různých podporovaných platformách (Android, iOS, Windows UWP). Zároveň však podporuje přizpůsobení stylu zvyklostem dané mobilní platformy – u Androidu se používá styl vycházející z Google Material Design (5), u iOS se vychází z iOS Human Style Gudelines (6).

Díky tomu se frameworku daří vyvarovat se chyb mnoha jiných, starších řešení, kde aplikace na mobilních telefonech nepůsobili přirozeně díky zcela jinému stylu ovládacích prvků a celkově výrazně jinému UX, než jaké je dané platformě zvyklé.

Framework Ionic se ve skutečnosti skládá z mnoha různých komponent, které propojuje dohromady a využívá:

1. HTML5 / CSS / JavaScript (webová aplikace)
2. Apache Cordova
3. Angular 5
4. Typescript / Node.js / Webpack
5. Crosswalk / Android WebView / WKWebView

Základem aplikací v Ionic jsou právě HTML 5 / CSS / JavaScript technologie – v podstatě se totiž jedná o webové aplikace běžící na mobilním telefonu. Tyto technologie se používají přímo i nepřímo – některé části aplikace se do nich pouze překládají z jiných

jazyků (např. TypeScript do JavaScript), jiné se mohou používat i přímo (např. CSS – i když u Ionic aplikací spíše využíváme překlad z jazyka SCSS).

Angular představuje moderní webový framework, který se nijak neváže na použití v mobilních aplikacích – naopak, jeho použití je běžné při tvorbě standardních webových aplikací mimo mobilní platformy (ale vč. např. mobilních webů). Angular je základním stavebním kamenem Ionic – v současné době jsou všechny jeho prvky implementovány jako Angular komponenty / služby (verze 3). Ve verzi 4 Ionic přechází na systém Stencil (7) postavený nad standardem Web Components (8).

Apache Cordova (9) tvoří základy frameworku, které nám umožní vytvořit mobilní aplikaci. Princip je velmi jednoduchý – Cordova vytvoří malou, nativní aplikaci (pomocí platformě závislého kódu), která po svém spuštění vytvoří okno obsahující prohlížeč, ve kterém se spustí samotná aplikace. Že se jedná o prohlížeč je pro uživatele nepostřehnutelné – nejsou zde žádné ovládací prvky, adresní řádek či cokoliv jiného – pouze to, co zobrazí samotná aplikace.

O zobrazení samotného HTML obsahu se stará právě Android WebView / WKWebView či Crosswalk. Jde o různé technologie používané na platformách pro zajištění co nejmodernějšího zobrazovacího jádra (což je kritické, Ionic využívá mnoha vlastností moderních prohlížečů). WKWebView (10) je moderní jádro postavené na WebKit, které je součástí platformy iOS. Android WebView je řešení pro novější systémy Android od verze 5.0 (starší systémy WebView obsahují též, nicméně je nepoužitelný z důvodu nemožnosti aktualizace (11)). Řešením pro tyto starší systémy je právě projekt Crosswalk od společnosti Intel. V podstatě se jedná o jádro prohlížeče Chrome (používané i v Android WebView) zabalené jako součást aplikace. Díky tomu máme k dispozici moderní zobrazovací jádro, nevýhodou je ovšem výrazné zvětšení velikosti aplikace (desítky MB).

3.1 Apache Cordova

Framework Apache Cordova není na poli mobilních aplikací žádným nováčkem. Jeho prvopočátky se datují až do roku 2009 (12), kdy vznikl jako původní experiment na konferenci iPhoneDevCamp v San Franciscu pod původním názvem PhoneGap.

Myšlenka byla velmi jednoduchá – jedním z velkých problémů tehdejšího vývoje mobilních aplikací byla velká fragmentace platforem – v té době se jednalo zejména o Android, iOS a Windows Phone. Pokud chtěl vývojář podporovat všechny tyto platformy, znamenalo to vyvíjet (a udržovat) tři separátní aplikace s oddělenými zdrojovými kódy. Není tudíž s podivem, že vývojáři začali hledat způsoby, jak zajistit multiplatformní vývoj aplikací a snížit tak náročnost vývoje.

Phonegap přišel s velmi jednoduchou myšlenkou, a to založit framework na již existujících společných multiplatformních technologiích - tedy HTML, CSS a JavaScript. Aplikace se tedy skládá ze dvou propojených částí - první je samotná nativní aplikace (specifická pro každou platformu), která obsahuje jedinou obrazovku. Zde se zobrazuje nativní webový prohlížeč dané platformy tak, aby uživatel nepoznal, že se jedná o prohlížeč, tudíž bez ovládacích prvků (tlačítko zpět, adresní řádek atd.). Zároveň je tato část integrační, aplikaci zpřístupňuje vybrané API z JavaScriptu (např. informace o inicializaci aplikace, událost tlačítka zpět, informace o zařízení). Toto JavaScript API je nicméně stejné pro všechny platformy, každá nativní část jej musí naimplementovat a tím zajišťujeme nezávislost JS na dané platformě. Tato část je automaticky vygenerována nástroji frameworku (typicky dnes utilita cordova, dříve utilita phonegap). Do této části se neočekává, že by programátor měl jakkoliv zasahovat, maximálně pro přizpůsobení některých metadat aplikačních descriptorů (pList soubory na iOS, Application Manifest na Android).

Druhá část je samotná aplikace, která využívá běžné webové technologie jako je HTML, CSS a JavaScript. Aplikace tedy nevyužívá (využívat ani nemůže) žádný platformě závislý kód. Veškerá práce s nativním prostředím se děje pomocí nativního kódu Apache Cordova. To ovšem neznamená, že je možné takovouto aplikaci spustit v běžném webovém prohlížeči – zde bude chybět rozšiřující JS API poskytnuté frameworkem Cordova. Tyto funkce jednoduše v webovém prohlížeči nejsou k dispozici. To samozřejmě může predstavovat problém pro ladění a vývoj. Je zcela jistě jednodušší vývoj provádět lokálně na počítači než na mobilním telefonu či v emulátoru. Z toho důvodu se do projektu Cordova přidala podpora platformy „browser", která emuluje API Cordovy (alespoň částečně) a umožňuje tak běh v prohlížeči, čehož je často využíváno pro vývoj.

3.1.1 Přijetí platformy

Cordova se již z počátku setkala s velmi vřelým přijetím u vývojářů. Ačkoliv se objevovaly hlasy, že aplikace využívající webové technologie nebudou nikdy tak kvalitní/rychlé, jako ty nativní (což může být částečně pravda), ani tyto potenciální problémy vývojáře neodradili, spíše naopak. Možnost vyvíjet aplikace multifplatformě byla pro vývojáře natolik lákavá, že překonala i nedostatky platformy. Je samozřejmě potřeba si říci, že Cordova není vhodná pro všechny druhy aplikací – systémové utility, aplikace vyžadující specifickou funkcionalitu dané platformy či velmi těsnou integraci s ní nepředstavují vhodné kandidáty. Takových aplikací však zřejmě není majorita.

O popularitě frameworku svědčí již od počátku několik věcí – jednak množství aplikací známých společností, které framework využili (např. Facebook, Wikipedia, Salesforce, IBM a mnoho dalších), či fakt, že původní vývojářskou společnost Nitobi v roce 2011 odkoupil gigant Adobe (13). Do této doby se též datuje další významný milník - uvolnění celého frameworku Phonegap jako otevřeného softwaru pod názvem Apache Cordova (jak jej známe nyní) a s tím související předání zdrojových kódů nadaci Apache (Apache Foundation).

To ovšem rozhodně neznamenalo, že Adobe ztratil o Phonegap / Cordovu zájem – spíše naopak. Otevření platformy znamenalo její velký rozvoj, kdokoliv se mohl (a dodnes může) zapojit do vývoje, v případě problémů podívat do zdrojových kódů platformy či si ji upravit podle vlastních potřeb. Od této doby je framework známý pod dvěma názvy – Apache Cordova (otevřená varianta spravovaná Apache Foundation) a Phonegap (komerční služba poskytovaná společností Adobe). Neznamenalo to ovšem rozdvojení platformy. Adobe se obecně místo na prodej frameworku zaměřil na prodej služeb a podpory kolem tohoto nově vzniklého, otevřeného softwaru, což se ukázalo zpětně jako velmi rozumné rozhodnutí, které dovolilo platformě růst a rozvíjet se. Pro vývojáře se navíc jednalo o velmi sympatický krok, protože použitím této technologie se nestanou závislí na jednom konkrétním dodavateli (u kterého může hrozit ukončení vývoje či změna podmínek).

3.1.2 Použití frameworku

Použití frameworku je velmi jednoduché, pomocí utility cordova vygenerujeme základní kostru aplikace a přidáme platformy, pro které chceme provádět překlad. Zároveň se založí jednoduchá kostra zdrojových kódů – v podstatě jeden HTML a JavaScript zdrojový soubor. Pokud naše aplikace nepoužívá Cordova API (resp. se bez něj umí obejít pomocí detekce a emulace), je možné je spustit i v prohlížeči (či pomocí platformy browser).

Teoreticky zde můžeme začít náš vývoj aplikace s pomocí čistého HTML5 / CSS3 / JS (ECMAScript 6), nicméně toto prostředí nám poskytne jen velmi málo předpřipravených věcí pro vývoj mobilní aplikace. Proto vzniklo mnoho různých frameworků, které nad těmito technologiemi staví a umožní nám řešit běžné úkoly v mobilním vývoji jako je např. navigace, rozdělení aplikace na sekce či ovládací prvky optimalizované pro mobilní zařízení. Jedním z takovýchto frameworků (a zřejmě na platformě Cordova nejpoužívanější) je právě Ionic.

3.1.3 Distribuce aplikace

Aplikace se distribuje stejně jako její nativní ekvivalenty – vznikne binární soubor, do kterého se aplikace přeloží a zabalí. Tento proces je plně automatizovaný pomocí utility cordova z frameworku. Překlad se provádí zvlášť pro každou platformu (máme jedny zdrojové kódy, více výstupních binárních aplikací).
Je zde třeba podotknout, že proces překladu a následné distribuce vyžaduje:

- Přidání platformy v projektu Cordova (díky tomu se vygeneruje nativní část),
- Instalaci vývojových nástrojů pro danou platformu (Android SDK, Xcode pro iOS),
- Podporovanou platformu pro vývoj (např. MacOS pro iOS aplikace).

Dále je nutné poznamenat, že pokud vývojář plánuje umístit aplikaci na veřejně dostupné obchody dané platformy (Google Play / Application Store), je nutné zřídit (a zaplatit) vývojářské účty na těchto službách. V tom se ale Cordova nijak neliší od vývoje a distribuce nativních aplikací.

3.2 SPA

Single Page Application (SPA) (14) je velmi důležitým konceptem dnešních webových (a tudíž i Cordova) aplikací. Standardní webové aplikace implementované v tzv. serverových frameworcích fungují pomocí následujícího schématu:

- HTTP požadavek na dané URI;
- Server vyhodnotí aktivní obsah (controller, šablona);
- Server odešle výsledné HTML klientovi.

Tento model je samozřejmě funkční, nicméně obsahuje mnoho problémů pro zajištění dobré uživatelské zkušenosti (User Experience - UX) plynoucí z původně úplně jiného zaměření webových technologií, např.:

- Bezestavovost HTTP protokolu (každý požadavek je nezávislý);
- Při každé navigaci se načítá celá stránka znovu (a ztratí se původní obsah);
- Datová neefektivita (výsledné HTML může být velké).

Všechny tyto problémy jsou samozřejmě řešitelné (minimálně do nějaké míry), ale může se jednat o komplikovaný proces.

Myšlenka SPA aplikací naproti tomu využívá vlatností moderních webových technologií a mění daný proces:

- HTTP požadavek na danou aplikaci;
- Do prohlížeče se načte celá aplikace;
- Spustí se JavaScript kód aplikace.

Další interakce v rámci aplikace je tudíž již na jedné jediné načtené stránce, jejíž obsah se mění na základě pokynů JavaScript kódu. Tímto způsobem tedy může aplikace emulovat chování i klasických aplikací jako je např. navigace, odkazy atd. Pro uživatele je vše transparentní; že se jedná o SPA pravděpodobně vůbec nepozná.

Tento způsob vývoje přináší velké výhody:

- aplikace se načte jednou a její kódy mohou být uloženy ve vyrovnávací paměti prohlížeče (cache) – opětovné načtení je rychlé nebo okamžité;
- se serverem se komunikuje pomocí REST služeb - načítají se tedy jen data, nikoliv jejich prezentace (efektivnější na přenos);
- aplikace si může velmi jednoduše lokálně uchovávat stav všech součástí;
- aplikace může např. komunikovat se serverem na pozadí, kontrolovat zda nedošlo

k vzniku událostí a prezentovat je uživateli;

- HTML 5 tyto možnosti ještě dále velmi výrazně vylepšuje – stačí zmínit technologie jako jsou webové notifikace (Web Notification API), možnost běhu na pozadí i po zavření okna (Service Worker), hardwarově akcelerované vykreslování grafiky (canvas pro 2D, WebGL pro 3D) a mnohé další.

Pro příklady SPA aplikací není nutné chodit daleko – setkáváme se s nimi denně, pokud jsme uživateli běžných internetových služeb – příkladem je např. Facebook, Youtube, Google Documents, Office 365 či všechny webové komunikátory (WhatsApp, Skype a další).

Je možné konstatovat, že SPA se stalo standardem pro vývoj webových aplikací a klasické serverové řešení zůstaly pouze u webových stránek (15).

3.3 Angular

Angular (16) je jedním z nejpopulárnějších frameworků pro vývoj SPA řešení. Pochází z dílny společnosti Google, která ho používá pro vývoj svých interních aplikací (např. Google AdWords). Spolu s (konkurenční) knihovnou React (17) od společnosti Facebook představují nejběžnější řešení pro současné SPA.

U tohoto řešení je potřeba rozdělit nicméně:

- AngularJS (nesprávně Angular 1.x);
- Angular (první verze nese označení 2, současná verze 5).

Ačkoliv se může jednat o na první pohled pouze o pokračování AngularJS, opak je pravdou, tyto frameworky mají jen velmi málo společného, Angular 2 byl napsán odpočátku se zcela novým návrhem architektury (18). V práci se budu zabývat pouze novějším řešením – tedy Angular 2+ (další verze představují pouze evoluční změny).

Angular je podobně jako React postaven na principu komponent. Každá komponenta je tvořena:

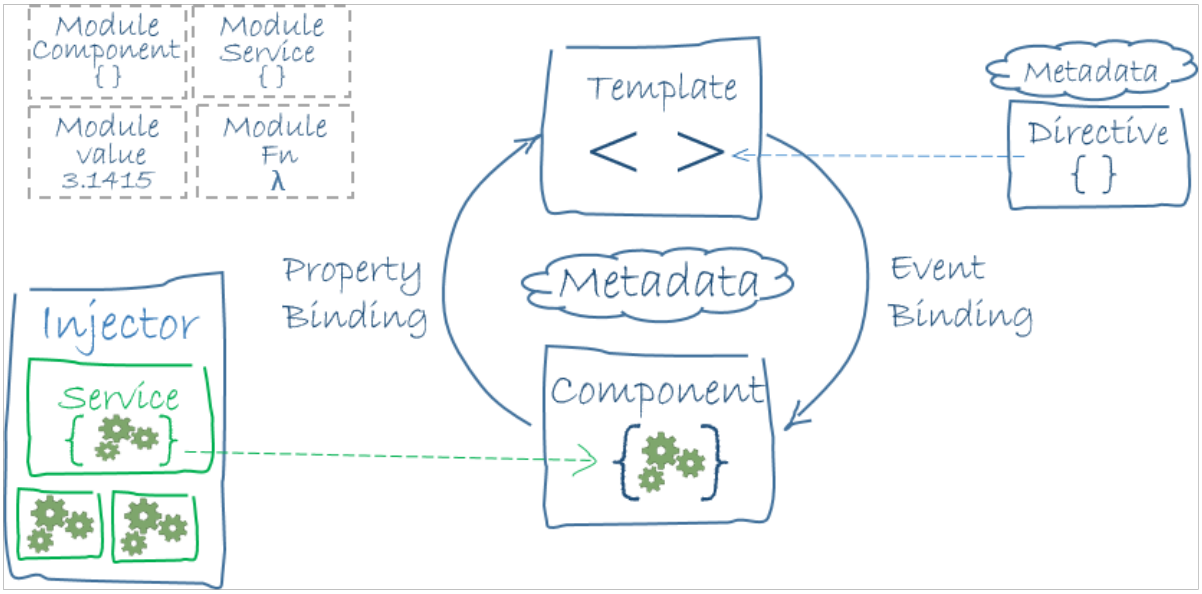
- šablonou (HTML);
- třídou (samotná komponenta).

Dohromady takto tvoří komponenta a šablona samostatný funkční celek, který může být

zobrazen uživateli a který s ní může pracovat. Myšlenka komponent je postavena na požadavku znovu použitelnosti kódu a zamezení duplikace funkcionality.

Dalšími klíčovými prvky Angularu jsou zejména:

- injector (mechanismus dependency injection);
- direktivy (generalizace komponent, může být použito pro implementování např. Podmínek);
- metadata komponent a direktiv;
- property binding a event binding.



Obrázek 1: Architektura aplikace v Angularu 2+. [cit. 2018-05-27]. Dostupný z: https://angular.io/generated/images/guide/architecture/overview2.png

3.3.1 Podporované programovací jazyky

Angular jako takový je v podstatě nezávislý na použitém programovacím jazyku, pokud je možné ho přeložit do JavaScriptu s příslušnými metadaty.

Mezi oficiálně podporované jazyky patří:

- TypeScript (nejpoužívanější);
- JavaScript;
- Dart.

Nejčastěji se setkáme s jazykem TypeScript (19) z dílny společnosti Microsoft. TypeScript (TS) je syntaktickou nadmnožinou jazyka JavaScript (konkrétně jeho verze ES6), ke kterému navíc přidává spoustu užitečných vlastností pro velké aplikace jako je např. typová kontrola v době překladu, podpora dekorátorů a další.

TS se vždy překládá do JavaScriptu, který následně běží v prohlížeči. Podporován je překlad i do velmi starých dialektů JS, nejčastěji se ovšem setkáme s překladem do ES 5 (rok 2009), který je podporován všemi dnes používanými prohlížeči perfektně. Za zmínku stojí i fakt, že samotný Angular je též naprogramován v jazyku TypeScript.

3.3.2 Mechanismus dependency injection

Dependency injection představuje aplikaci návrhového vzoru obrácení řízení (Inversion of Control) do mnoha řešení vč. Angularu. Je postaveno na myšlence, že komponenta místo toho, aby své závislosti vytvářela sama, tak se o správu jejich životního cyklu (vytváření, zánik) stará prostředí / framework.

Tento mechanismus je velmi široce používaný v mnoha řešení - namátkou můžeme zmínit např. backendové frameworky Java EE, Spring, ASP.NET Core, PHP Symphony a mnohé další.

V Angularu se s tímto mechanismem setkáme na každém kroku – pokaždé, když budeme chtít použít službu poskytovanou frameworkem (či vlastní službu), získáme její instanci pomocí právě pomocí dependency injection.

3.3.3 Propojení vlastností a událostí

Propojení vlastností (property binding) a událostí (event binding) představují zcela klíčovou vlastnost platformy Angular a zřejmě nejzásadnější rozdíl od předchozí verze platformy (AngularJS). Zatímco AngularJS používal koncept dvoucestného bindingu (two way data binding), Angular striktně používá jednocestný property binding a event binding.

Property binding: Umožňuje nám definovat automatické propojení vlastnosti komponenty na jiném výrazu / vlastnosti komponenty nadřazené. Příkladem může být

např. propojení vlastnosti „uživatel" komponenty „Odhlášení" s vlastností „uživatel" aplikační komponenty. Do podřízené komponenty se tedy vždy automaticky předá aktuální hodnota uživatele z aplikační komponenty. Pokud dojde ke změně v rodičovské komponentě, změna se automaticky promítne i v propojené komponentě. Obecně můžeme říci, že se jedná o propojení směrem „do komponenty" (předáváme jí tímto data). V HTML šabloně toto propojení zapisujeme pomocí znaků [property].

Event binding: Toto propojení umožňuje komponentě generovat události, kterým naslouchá nadřízená komponenta (volitelně). Příkladem takovýchto propojení může být např. událost „click" na tlačítku - rodičovská komponenta se díky němu dozví o tom, že uživatel na tlačítko kliknul. Jedná se tedy o směr z komponenty - ta pomocí něj informuje, že došlo k nějaké situaci či změně. V šabloně zapisujeme jako (event).

3.3.4 Obousměrné propojení

Nemožnost obousměrného propojení byla jednou z důležitých architektonických rozhodnutí při vývoji platformy Angular, po vzoru platformy React, kde byla možnost obousměrného propojení též vyloučena.

Toto řešení má řadu výhod, které jsou nejlépe vidět při srovnání se starším AngularJS – zde obousměrné propojení způsobovalo spoustu problémů s výkonem aplikace, jelikož bylo neustále nutné sledovat změnu na obou stranách propojení.

Na druhou stranu existuje spousta situací, kde je obousměrné propojení výhodné, ba dokonce nutné. Klasickým příkladem může být např. obyčejný input tag v HTML, který je při otevření naplněn z komponenty do šablony a následně při změně uživatelem je nutné promítnout změny dat zpět do komponenty.

3.4 Ionic

Ionic framework (20) je v současné době (verze 3) postavený na přímo na frameworku Angular. Budoucí verze (4, v současné době ve vývoji) integruje standard Web Components, na Angularu je nezávislý, i když se předpokládá jeho majoritní použití vývojáři. Budoucí verze nemá Angular v žádném případě odstranit, či odsunout na druhou kolej, cílem změny je naopak umožnit použití Ionic frameworku i vývojářům preferujících jiné technologie – např. React či VueJS.

Angular jako takový je zcela nezávislý na použití v rámci konkrétní technologie – jediné, co potřebuje, je moderní webový prohlížeč. Poskytuje programátorovi základní funkce nutné pro vývoj aplikace – možnost tvorby komponent, routování URL, property/event binding a dependency injection. Neposkytuje nám ale žádné komponenty pro tvorbu uživatelského rozhraní, nenabízí žádnou integraci s Cordova, jedná se jednoduše o webový framework pro vývoj aplikací, nic více, nic méně.

A zde přesně přichází Ionic, který nám poskytuje několik zcela zásadních vlastností pro vývoj mobilních aplikací:

- napojení na framework Cordova;
- komponenty uživatelského rozhraní podporující dotykové ovládání;
- integraci s nativní funkcionalitou mobilních telefonů;
- podpůrné nástroje a utility pro sestavení.

Díky tomu máme k dispozici technologii, která nám umožňuje z pohledu výojáře snadno propojit vývoj v Angularu, ale zároveň nám doplní předpřipravenou integraci s Cordova frameworkem. Důležité je zde zmínit, že Ionic technologii Angular nenahrazuje, ale naopak – vychází z ní a rozšiřuje o další možnosti.

3.4.1 GUI komponenty

Ionic poskytuje velké množství GUI komponent optimalizovaných pro použití na mobilních zařízeních jako je např. seznam, navigační menu, formulářové vstupní prvky, toast (informační zpráva zobrazená krátkou dobu na zařízení). Tyto komponenty jsou implementovány jako komponenty technologie Angular, tudíž pro jejich použití nemusíme používat nic speciálního či specifického pro Ionic. Všechny komponenty podporují samozřejmě i property či event binding, jejich použití v Angularu je tedy zcela přirozené.

3.4.1.1 Vzhled komponent

Jednou z velkých přínosů Ionic frameworku oproti starším technologím (např. jQuery Mobile) je přizpůsobení vzhledu komponent dané platformě. Ionic využívá tzv. „Platform CSS" – tedy specifické CSS styly, které se aplikují pouze na určitých platformách. Cílem je zajistit pokud možno co nejpodobnější vzhled k nativním aplikacím – tzn. aby pro

uživatele dané platformy vypadaly přirozeně.

Ionic standardně používá následující platformní vzhledy:

- Material Design – používaný na platformě Android, postavený na stejnojmeném designovém vzoru společnosti Google,
- iOS Design – standardně využitý na platformě iOS společnosti Apple, vycházející z jeho standardního vzhledu a iOS Design Guidelines (21),
- Windows Phone design pro použití na mobilní platformě společnosti Microsoft, od jeho dalšího rozvoje se nicméně ustupuje z důvodu zániku této platformy.

3.4.1.2 Responzivní design

Responzivní design (22) je moderním trendem ve vývoji webu a návrhu uživatelských rozhraní – základní myšlenka je velmi jednoduchá – design se přizpůsobí velikosti obrazovky tak, aby byl stejný design použitelný na různých rozlišeních a fyzických velikostech obrazovky.

Tento přístup je při vývoji multiplatformních aplikací nesmírně důležitý – aplikace může běžet na mobilním telefonu (a každý model má jiné rozlišení a obrazovku), ale stejně tak i např. na tabletu či tablet PC.

Responzivní design má několik základních odlišností oproti dříve používaným postupům:

- nepoužívání absolutních souřadnic;
- omezené používání absolutních jednotek (např. pixel) - preferujeme relativní (např. vw / vh - poměr k šířce / výšce stránky, em / rem - vychází z velikosti písma);
- flexibilní změna layoutu v závislosti na velikosti (např. na velkém displayi zobrazíme menu vždy, na malém ho schováme atd.).

Klíčovými prvky pro implementaci responzivního designu jsou následující CSS3 technologie:

- Flexbox,
- Grid.

Flexbox (23) je technologie používaná pro dynamické, horizontální pozicování. Používá se

např. na nastavení velikostí ovládacích prvků vedle sebe (např. pro formuláře).

Grid (24) naproti tomu rozšiřuje možnosti Flexbox o možnost vertikálního pozicování. Obě technologie umožňují definovat, které prvky se mohou zvětšovat a jak, stejně tak definovat jejich rozměry relativně vůči jiným ovládacím prvkům.

V Ionic frameworku jsou obě dvě technologie využity pro implementaci tzv. typografické mřížky. Její princip je velmi jednoduchý – horizontální plocha je rozdělena na 12 samostatných částí a velikosti jednotlivých komponent specifikujeme jako násobky jednoho dílku.

Druhým neméně důležitým principem jsou tzv kategorie zařízení (velikosti displayů) (10) :

- xs
- sm - 576px
- md - 768px
- lg - 992px
- xl - 1200px

Daná kategorie se aplikuje tehdy, pokud je šířka plochy větší nebo rovna specifikované hraniční (tudíž např. lg se použije pro displaye o šířce větší, nebo rovné 992px).

Tyto kategorie jsou využity ve spolupráci s Flexbox a Grid k vytvoření různých velikostí pro různé displaye - např. pro lg 1/12, ale pro sm např. 4/12 šířky mřížky. Grid se automaticky navíc postará o zalomení v případě, že ve velikost v jednom řádku by přesáhla 12 / 12 velikosti.

To nám umožňuje velmi efektivně vytvářet uživatelská rozhraní, která vypadají dobře jak na velkém displayi (PC, tablet), tak malém (mobilní zařízení).

3.4.2 Programové komponenty

Programové komponenty technologie Ionic jsou primárně klasickými Angular službami – o jejich vytváření se tedy stará přímo jeho jádro a získání probíhá pomocí Injectoru

(DI mechanismus). Krom toho některé komponenty mají vazby přímo na API poskytované Cordovou – implementují tedy funkcionalitu vázající se na mobilní prostředí.

3.4.3 Zásuvné moduly

Ionic podporuje velké množství zásuvných modulů (pluginů) (25). Tyto pluginy se typicky váží na Cordova pluginy. Důvodem tohoto "zabalení" je to, že Cordova používá čistý JavaScript. V Ionic frameworku nicméně programujeme v jazyce TypeScript, který je synaktickou nadmnožinou JavaScript / ECMAScript 6.

Přímé použití Cordova pluginů by tedy pro programátora nebylo pohodlné, ani přirozené. Právě z tohoto důvodu se využívají zásuvné moduly Ionic, které do aplikace typicky naimportujeme jako Angular moduly. Poskytuje TypeScript / Angular služby, které můžeme získat pomocí DI. Tyto služby neprovádí většinou žádnou vlastní logiku, ale pouze provolávají příslušné Cordova moduly a poskytují typové definice jejich datových struktur.

Příklad takovýchto Ionic pluginů je např:

- Keyboard (ovládání zobrazení klávesnice);
- Storage (přístup k datovému úložišti telefonu);
- FCM (integrace s Google Firebase - Push Notifikace).

Celkové množství Ionic pluginů je obrovské – jejich několik set. Pro drtivou většinu běžně používaných Cordova pluginů nalezneme jejich odpovídající integraci ve formě oficiálně podporovaného Ionic pluginu.

3.4.4 Podpůrné nástroje

Ionic tým vyvinul nástroj „ionic-cli" (26), podpůrnou utilitu pro příkazovou řádku, pro zakládání nových projektů, překlad existujících a správu pluginů. Jedná se o alternativu k utilitě „ng" poskytované přímo frameworkem Angular, která slouží pro sestavování a testování aplikací. Tato utilita (ionic-cli) integruje několik pomocných technologií pro zajištění své funkcionality:

- npm (NodeJS Package Manager) – správa závislostí projektu;
- WebPack – sestavení výsledné aplikace, minifikace JS a CSS souborů;

- Cordova – správa platformních aplikací a jejich pluginů;
- tsc (TypeScript Compiler) – překlad TypeScript kódů do ECMASCript 2015;
- Sass (Syntactically Awesome Style Sheets) – překlad SCSS stylů do CSS;
- Ionic Pro – cloudová volitelná komerční služba pro testování a sestavování aplikací poskytovaná tvůrci frameworku.

Použití ionic-cli tedy není v žádném případě nutné či vyžadované – na druhou stranu integrace těchto všech (nutných) technologií dohromady je velmi složitá a komplikovaná. Tudíž použití ionic-cli je naprostým standardem snad u všech projektů využívajících Ionic 3.

Ve verzi 4 nás nicméně čeká užší integrace s utilitou „ng" od Angular týmu – důvod je zcela jednoduchý, vývoj obou nástrojů je velmi náročný a v současné době řeší ionic-cli spoustu věcí, které se již podařilo vyřešit velmi dobře v „ng".

4 Praktický úvod do vývoje v Ionic frameworku

V této kapitole si popíšeme praktický úvod do programování mobilních aplikací ve frameworku Ionic.

4.1 Příprava projektu a prostředí

Než se pustíme do samotného vývoje, je nutné provést několik základních kroků – připravit si prostředí pro vývoj a založit nový Ionic projekt.

4.1.1 Instalace prostředí

Pro vývoj aplikace budeme potřebovat několik základních nástrojů:

- NodeJS (www.nodejs.org) – LTS verze
- Platformní SDK:
  - Google Android:
    - Java 8 či vyšší (JDK)
    - Android SDK (https://developer.android.com/studio/)
    - Maven (https://maven.apache.org)
  - Apple iOS:

- Xcode (k dispozici z Apple AppStore)

Vývoj aplikací pro Android je možné na OS Widows, Linux a MacOS. Pro ladění iOS verze nicméně můžeme použít pouze operační systém MacOS, budeme tedy potřebovat počítač od společnosti Apple.

Po instalaci nezbytných prerekvizit můžeme přistoupit k instalaci základních nástrojů Ionic z příkazové řádky (je nutné mít práva administrátora):

```
$ npm install -g @ionic/cli
```

4.1.2 Založení nového projektu

Nový projekt můžeme založit pomocí utility ionic-cli:

```
$ ionic new <složka s projektem>
```

Utilita se nás zeptá na výběr šablony projektu – ta může být buď blank (prázdný projekt), nebo můžeme využít některou z předpřipravených příkladů (27).

4.2 Práce s projektem

Při vývoji můžeme aplikaci testovat dvěma různými způsoby:

- v prohlížeči;
- na zařízení.

Oba způsoby mají své výhody ale i nevýhody. Vývoj a testování v prohlížeči (jakýkoliv moderní prohlížeč – Google Chrome, Safari, Firefox či Microsoft Edge je podporován) je jednoznačně rychlejší – své změny vidíme okamžitě, vývoj je velmi flexibilní. Při volbě prohlížeče je možné nicméně doporučit buď Google Chrome, nebo Safari. Chrome používá stejné vykreslovací jádro (Blink) jako Android WebView (či Crosswalk), Safari zase používá jádro WebKit, které nativně využívá WKWebView na platformě iOS.

Ačkoliv by samozřejmě náš kód měl být zcela přenositelný bez využívání nestandardních rozšíření, praxe nám ale jasně říká, že testování na co nejpodobnějším vykreslovacím jádře

a JS implementaci k těm, na kterých aplikace v důsledku poběží na mobilním telefonu, jednoznačně zefektivňuje naši práci. Je nutné si ale uvědomit, že testování na prohlížeči nikdy není dostatečné – aplikaci musíme vždy zkontrolovat i na zařízení. Testování na prohlížeči je pouze cestou jak si zjednodušit vývoj před finálním laděním, které vždy musí proběhnout na mobilním telefonu či tabletu (a ideálně na několika reprezentativních modelech – různé modely, verze systému atd.). Ačkoliv je Ionic multiplatformní, rozdíly mezi platformami se samozřejmě vyskytují, stejně jako rozdíly implementací některých funkcí v JS, CSS atd.

Dále je nutné mít na paměti, že v prohlížeči nám nikdy nebudou fungovat žádné Ionic pluginy. Pokud chceme tedy ladění v prohlížeči využívat, aplikace s tímto musí počítat a čím více je na použití pluginů závislá, tím méně použitelné pro nás bude používat prohlížeč pro vývoj.

Velkou výhodou ladění v prohlížeči je též dostupnost vývojářské konzole prohlížeče, kde si můžeme snadno vypisovat ladící zprávy, sledovat stav programu či ho krokovat (debuggovat). Tyto vývojářské nástroje můžeme omezeně použít i v případě ladění na zařízení na podporovaném prohlížeči (Google Chrome).

Ladění na mobilním zařízení je tedy mnohem více průkaznější, nicméně má i své nevýhody. V první řadě je celý proces výrazně pomalejší, jelikož pro nasazení nové verze na mobilní telefon je nutné:

1. provést kompilaci aplikace do JS / CSS balíků;
2. přeložit nativní platformní kostru aplikace;
3. sestavit výsledný balík (APK či IPA podle platformy);
4. nahrát aplikaci na mobilní telefon.

Při ladění v prohlížeči se provádí pouze první krok, navíc na počítači, takže je celý proces pochopitelně mnohem rychlejší.

4.2.1 Spuštění vývojového serveru

Vývojový server můžeme spustit pomocí následujícího příkazu ze složky projektu (tam,

kde máme soubor package.json se závislostmi):

```
$ ionic serve
```

Aplikace bude k dispozici na webové adrese http://localhost:8100 (Ionic 3) či http://localhost:4200 (Ionic 4).

4.2.2 Spuštění aplikace na mobilním zařízení

Abychom mohli spustit aplikaci na mobilním zařízení, je nejprve nutné připojit zařízení pomocí datového kabelu k počítači a provést spárování. Tento postup se liší podle platformy, stejně jako jednotlivé příkazy pro Ionic.

4.2.2.1 iOS

Pro připojení telefonu či tabletu (iPhone či iPad) musíme spustit aplikace iTunes (pokud se tak nestane automaticky) a na zařízení potvrdit důvěryhodnost daného počítače.

Poté můžeme spustit projekt pomocí následujícího příkazu:

```
$ ionic cordova run ios
```

Aplikace by se měla přeložit (první překlad trvá déle díky připravě platformního projektu) a automaticky nahrát na mobilní zařízení. Pokud se tomu tak nestane, je nutné otevřít Xcode a otevřít v něm platformní projekt, která najdeme ve složce platforms/ios. Nejčastějším důvodem selhání bude problém s podepisovacími certifikáty a „provisioning profiles", příp. nenastavení vývojového týmu.

V Xcode musíme být přihlášení pomocí našeho iTunes Connect účtu, mít zaplacený vývojářsky účet (či být členy vývojářského týmu v iTunes Connect) a mít zapnutou automatickou správu „provisioning profiles". Pak se Xcode automaticky postará o jejich případné vygenerování a automatické podepsání aplikace pro vývoj.

V případě, že provisoning profiles potřebujeme přegenerovat (např. při změně UDID přes Apple Developer Portal), je nutné smazat obsah složky ~/Library/MobileDevice/Provisioning Profiles/.

4.2.2.2 Android

Prvním nutným krokem je aktivace vývojářského režimu na telefonu a povolení připojení ADB přes rozhraní USB. Bohužel, tento postup se liší podle verze systému i konkrétního modelu zařízení. U většiny telefonů a tabletu je nicméně možné aktivovat vývojářský režim rychlým tapnutím, 7x za sebou, na číslo sestavení v menu nastavení / O telefonu. Tím se ve většině případů aktivuje i připojení přes USB, konkrétní postup je ale závislý na telefonu.

Před spuštěním projektu je nutné ověřit propojení telefonu a Android SDK. Pro většinu mobilních telefonů není potřeba žádný speciální postup, výjimku tvoří telefony značky Samsung, které vyžadují instalaci zvláštního ADB (Android Debug Bridge) ovladače. Při prvním připojení k počítači budeme vyzvání na ověření jeho certifikátu na mobilním zařízení, což je nutné schválit.

Funkčnost ADB rozhraní můžeme snadno ověřit pomocí stejnojmeného příkazu:

```
$ adb devices -l
```

Tento příkaz nám vypíše připojená zařízení a naše zařízení tam musíme vidět.

Následně můžeme spustit projekt pomocí následujícího příkazu ze složky projektu:

```
$ ionic cordova run android
```

4.3 Základy vývoje

V této kapitole se popisují základní aspekty vývoje v technologii Ionic.

4.3.1 Stránky

Stránka je komponentou, která může být použita v navigaci. Stránky jsou tedy kořenovými (top level) komponentami, které uživatel vidí. Stránka stejně jako každá jiná komponenta obsahuje šablonu a třídu.

Stránku můžeme vytvořit manuálně, nebo pomocí uitility ionic-cli:

```
$ ionic g page <název stránky>
```

Tím se automaticky vygeneruje prázdná třída s přislušnou anotací (@Component), prázdná šablona (HTML), styl (Sass) a jednotkový test (spec).

Po vygenerování stránky je ovšem nutné, pokud ji chceme používat v aplikaci, zařadit tuto novou komponentu do některého modulu (příp. aplikačního modulu). V definici modulu musíme přidat referenci na stránku do následujících částí:

- entryComponents,
- imports

4.3.2 Služby

Služby jsou třídy implementované v jazyce TypeScript, jejichž instance můžeme získat pomocí mechanismu dependency injection. Tyto instance jsou tzv. singletony – tzn. existuje vždy maximálně jedna instance, která je sdílená mezi jednotlivými komponentami. Služby tedy můžeme použít i pro předávání dat mezi komponentami. Těmto vlastnostem musíme ale samozřejmě přizpůsobit i její návrh, musíme počítat s přístupem z nezávislých komponent (nikoliv paralelním, ale asynchronním – což ovšem plyne z návrhu jazyka, nikoliv samotného Ionic frameworku).

Vygenerovat službu můžeme pomocí příkazu:

```
$ ionic g service <název třídy>
```

Tato třída musí být importována v modulu v sekci imports.

5 Závěr

V rámci proběhnuvšího školení došlo k ověření metodických postupů na skupině programátorů s různými vstupními znalostmi a zvolený postup se ukázal jako úspěšný. Zejména vhodné se ukázalo praktické demonstrování probíraných technologií a postupů, které si účastníci mohli okamžitě vyzkoušet (a vidět je na svých mobilních zařízeních). Teoretické části výkladu se ukázali jako více problematické, u delšího výkladu byl problém udržet pozornost účastníků, proto jsem se rozhodla je v maximální míře zkracovat – ale tak, aby zůstala zachována logika školení, spíše jsem se snažila zkrátit jednotlivé teoretické části a rovnou je demonstrovat v rámci praktické ukázky. Tento postup se osvědčil, účastníci byli schopni lépe udržet pozornost a nové poznatky lépe akceptovali.

V rámci své práce jsem vytvořila pomocné školící materiály – např. prezentační slajdy, které byly použity zejména v případě teoretického výkladu. V praxi se osvědčily, účastníci je považovali za vhodné pro lepší pochopení problematiky.

Obecně měli účastníci často problémy s pochopením celkové architektury frameworku díky velkému množství použitých technologií (ionic-cli, Angular, Ionic, Webpack, Cordova a další), též se základními operacemi v projektu (i když např. pochopili teoretické koncepty), zejména tam, kde museli používat pomocné utility (ionic-cli) pro základní operace typu generování komponenty či služby, proto jsem se rozhodla tyto nejčastější problémy zpracovat speciálních materiálů (studijní opora).

Při hodnocení a poskytování zpětné vazby jednotlivých samostatných a kontrolních prací se ukázalo zcela klíčové poskytnutí individuálního přístupu. U zpětné vazby umožnit studentům klást otázky, individuálně zhodnotit jejich postup, poskytnout rady pro další postup. U kontrolních prací potom dát prostor studentům své řešení obhájit, byť je chybné. Pro pochopení je klíčové, aby porozuměli svým vlastním chybám a nebrali názor lektora jako fakt, ale skutečně viděli objektivní důvody, proč je jimi navrhované řešení lepší a vhodnější.

V budoucnosti bych ráda tuto studijní oporu chtěla rozšířit i o další kapitoly – na to je ovšem nutné získat další data a zpětnou vazbu od studentů, jinak by se z ní velmi jednoduše mohla stát referenční příručka, což dle zpětné vazby studentů je zcela kontraproduktivní (ztrácí se v ní, stejně jako se ztráceli v oficiální, byť velmi neúplné anglické dokumentaci - je pro ně příliš konkrétní, podrobná, málo zaměřená na popis problémů a „hands on" přístup, místo toho popisuje do detailu každou nuanci té či oné části).

Další rozvoj bude nutný i z důvodu rozvoje frameworku jako takového - v současné době je ve vývoji verze Ionic 4, která přináší několik zásadních změn (odpoutání od Angular, přechod na WebComponents), nicméně zatím není k dispozici ve finální podobě, tudíž není k dispozici ani postup pro migraci na tuto novou verzi. Výjadření vývojářů nicméně říká, že základní principy a použité technologie zůstanou zachovány, budou se měnit hlavně detaily použití jednotlivých komponent.

Reference

1. PAMELA R. FERENCE, Edward L. Vockell. Adult Learning Characteristics and Effective Software Instruction. *EDUCATIONAL TECHNOLOGY* [online]. July-August 1994, **34**(6), 25–31 [vid. 2018-05-09]. Dostupné z: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.473.9293&rep=rep1&type=pdf

2. HENG, Seah Kar, Irina ARISTARKHOVA, Stan JARZABEK a Pin-Kwang ENG. Combining Theory and Practice in the Right Proportions. *CDTL Brief* [online]. 2005, **8**(4), 6 [vid. 2018-05-27]. Dostupné z: http://www.cdtl.nus.edu.sg/brief/Pdf/v8n4.pdf

3. WILSON, Karen a James H. KORN. Attention During Lectures: Beyond Ten Minutes. *Teaching of psychology* [online]. 2007, **34**(2), 85–89. ISSN 0098-6283. Dostupné z: doi:10.1080/00986280701291291

4. *What Makes for Effective Adult Learning* [online]. B.m.: Wisconsin Union. [vid. 2018-05-09]. Dostupné z: https://union.wisc.edu/assets/Uploads/Events-Activities/Wheelhouse-Studios/MiniCourses-EffectiveLearning.pdf

5. Design. *Material Design* [online]. [vid. 2018-05-08]. Dostupné z: https://material.io/guidelines/

6. Human Interface Guidelines. *Designing for iPhone X* [online]. [vid. 2018-05-08]. Dostupné z: https://developer.apple.com/design/

7. *Stencil* [online]. [vid. 2018-05-08]. Dostupné z: https://stenciljs.com/

8. *webcomponents.org - Discuss & share web components* [online]. [vid. 2018-05-08]. Dostupné z: https://www.webcomponents.org/

9. *Apache Cordova* [online]. [vid. 2018-05-27]. Dostupné z: https://cordova.apache.org/

10. Grid. *Ionic Framework* [online]. [vid. 2018-05-27]. Dostupné z: https://ionicframework.com/docs/api/components/grid/Grid/#default-breakpoints

11. *WebView for Android - Google Chrome* [online]. [vid. 2018-05-08]. Dostupné z: https://developer.chrome.com/multidevice/webview/overview

12. TAFT, Darryl K. PhoneGap Simplifies iPhone, Android, BlackBerry Development. *eWEEK* [online]. 13. březen 2009 [vid. 2018-05-08]. Dostupné z: http://www.eweek.com/development/phonegap-simplifies-iphone-android-blackberry-development

13. SHANKLAND, Stephen. Adobe buys PhoneGap, TypeKit for better Web tools. *CNET* [online]. 3. říjen 2011 [vid. 2018-05-08]. Dostupné z: https://www.cnet.com/news/adobe-buys-phonegap-typekit-for-better-web-tools/

14. Single-page Applications. *Code School* [online]. [vid. 2018-05-27]. Dostupné z: https://www.codeschool.com/beginners-guide-to-web-development/single-page-applications

15. SKÓLSKI, Paweł. Single-Page Application vs. Multiple-Page Application - Neoteric. *Neoteric* [online]. 1. prosinec 2016 [vid. 2018-05-27]. Dostupné z: https://neoteric.eu/single-page-application-vs-multiple-page-application

16. *Angular Docs* [online]. [vid. 2018-05-27]. Dostupné z: https://angular.io/

17. *React - A JavaScript library for building user interfaces* [online]. [vid. 2018-05-27]. Dostupné z: https://reactjs.org/index.html

18. AngularJS Vs. Angular 2 Vs. Angular 4: Understanding the Differences. *Simplilearn.com* [online]. 27. únor 2018 [vid. 2018-05-27]. Dostupné z: https://www.simplilearn.com/angularjs-vs-angular-2-vs-angular-4-differences-article

19. *TypeScript - JavaScript that scales* [online]. [vid. 2018-05-27]. Dostupné z: https://www.typescriptlang.org/

20. IONIC. Build Amazing Native Apps and Progressive Web Apps with Ionic Framework and Angular. *Ionic Framework* [online]. [vid. 2018-05-27]. Dostupné z: https://ionicframework.com/

21. *Human Interface Guidelines* [online]. [vid. 2018-05-27]. Dostupné z: https://developer.apple.com/design/

22. *HTML Responsive Web Design* [online]. [vid. 2018-05-27]. Dostupné z: https://www.w3schools.com/html/html_responsive.asp

23. A Complete Guide to Flexbox | CSS-Tricks. *CSS-Tricks* [online]. [vid. 2018-05-27]. Dostupné z: https://css-tricks.com/snippets/css/a-guide-to-flexbox/

25. A Complete Guide to Grid | CSS-Tricks. *CSS-Tricks* [online]. [vid. 2018-05-27]. Dostupné z: https://css-tricks.com/snippets/css/complete-guide-grid/

25. Plugins. *Ionic Marketplace* [online]. [vid. 2018-05-27]. Dostupné z: https://market.ionicframework.com/plugins

26. Ionic CLI. *Ionic Framework* [online]. [vid. 2018-05-27]. Dostupné z: https://ionicframework.com/docs/cli/

27. Starters. *Ionic Marketplace* [online]. [vid. 2018-05-08]. Dostupné z: https://market.ionicframework.com/starters/

A Přílohy

A.1 Obsah elektronické přílohy

- Angular1 – projekt se zdrojovými kódy jednoduché aplikace ve frameworku Angular. Ukazuje příklad použití modulu, zapouzdření komponent, routování a práci s prostředím.
- Angular2 – komplexnější příklad aplikace s navigací, který se zaměřuje na interakci jednotlivých komponent, tvorbu služeb, vyvolání a zpracování událostí a použití knihoven třetích stran.