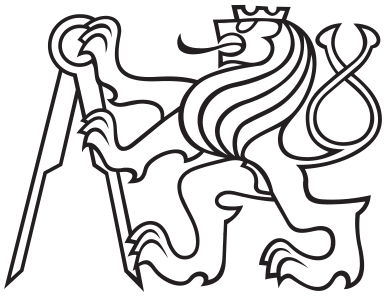
České vysoké učení technické v Praze
Fakulta elektrotechnická
Katedra počítačové grafiky a interakce



Diplomová práce

UX Eval - Webová aplikace pro analýzu testů použitelnosti

Bc. Dmitrij Bučkovský

Vedoucí práce: Ing. Ivo Malý, Ph.D.

Studijní program: Otevřená informatika, Magisterský

20. 5. 2018

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Bučkovský** Jméno: **Dmitrij** Osobní číslo: **406269**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačové grafiky a interakce**

Studijní program: **Otevřená informatika**

Studijní obor: **Interakce člověka s počítačem**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

UX Eval - Webová aplikace pro analýzu testů použitelnosti

Název diplomové práce anglicky:

UX Eval - Web application for usability test analysis

Pokyny pro vypracování:

Proveďte analýzu potřeb analytiků při testech použitelnosti. Zaměřte se zejména na způsob sběru dat, jejich dostupnost a použití. Dále proveďte analýzu existujících aplikací pro sběr a vyhodnocování dat z testů použitelnosti, zejména aplikace UX Tester a IVE tool. Navrhněte aplikaci, která umožní práci s projekty vzniklými v aplikaci UX Tester. Aplikace se zaměří na zobrazování dat v průběhu testu a při vyhodnocování testu. Pro aplikaci navrhněte uživatelské rozhraní a vytvořte ověřovací implementaci stěžejních funkcí, jako je například práce s textem, videem a připojení k projektům ve Firebase. Aplikaci implementujte ve webovém frameworku Angular. Výslednou aplikaci průběžně testujte jak z hlediska testů použitelnosti, tak z hlediska softwarového testování.

Seznam doporučené literatury:

M. Jones, G. Marsden, Mobile Interaction Design, Wiley, 2006
I. Malý, Analysis of Usability Tests with Context Model, Praha: 2012. PhD Thesis. České vysoké učení technické v Praze, Fakulta elektrotechnická.
J. Špatný, Mobile applications for management and data collections in field usability studies, Praha: 2017. Master Thesis. Czech Technical University in Prague, Faculty of Electrical Engineering.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Ivo Malý, Ph.D., katedra počítačové grafiky a interakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **07.02.2018** Termín odevzdání diplomové práce: **25.05.2018**

Platnost zadání diplomové práce: **30.09.2019**

Ing. Ivo Malý, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

.	
Datum převzetí zadání	Podpis studenta

Poděkování

Chtěl bych poděkovat svému vedoucímu Ing. Ivovi Malému, Ph.D. za cenné rady, pomoc a čas, který mi věnoval při psaní této diplomové práce.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze dne 20. 5. 2018 .....

Abstract

The goal of this work is to create web application, that will help researchers analyze data collected from usability tests. The first part of this work describes the meaning, process and way of collecting data from usability tests. In the second part, the web application is designed and then implemented using Angular 5 framework and TypeScript. Data is stored in the Firebase Realtime Database. In the end, the web application is tested by potential users.

Keywords

Usability testing, Angular, Firebase

Abstrakt

Cílem této práce je vytvořit webovou aplikaci, která bude pomáhat výzkumníkům analyzovat data shromážděná z testů použitelností. První část této práce popisuje význam, průběh a způsob sběru dat z testů použitelností. V druhé části je webová aplikace navržena, a potom implementována pomocí frameworku Angular 5 a TypeScript. Data se ukládají do Firebase Realtime databáze. Nakonec je aplikace testována potenciálními uživateli.

Klíčová slova

Test použitelnosti, Angular, Firebase

Obsah

Seznam obrázků

Kapitola 1

Úvod

Během vývoje aplikace nebo nového produktu je vhodné produkt průběžně testovat. Věci, které jsou vývojářům jasné a samozřejmé, nemusí být srozumitelné pro koncové uživatele. Proto se během vývoje produktu provádí testy použitelnosti na potenciálních koncových uživatelích.

Pro test použitelnosti si stanovíme úkoly, které chceme testovat a seženeme vyhovující participanty, nejlépe budoucí zákazníky, uživatele. Připravíme prostředí, které je co nejvíce podobné reálnému prostředí, a pomůcky, kterými budeme sbírat data. Největším zdrojem informací je video, proto budeme chtít zaznamenávat uživatelovou řeč těla, výraz tváře nebo například pohyb očí po obrazovce nebo pohyb myši, jakou klávesu participant zmáčkl nebo na jaké tlačítko klikl. Důležité je také i audio, participantovy komentáře nebo myšlenkové pochody (think aloud). Pozorovatelé si taky píši poznámky s časovými značkami. Po skončení testu dojde k analýze nasbíraných dat, výsledky se vyhodnotí a vyvodí se důležité poznatky, podle kterých se pokračuje ve vývoji nebo se opraví nalezené chyby.

K jednoduššímu sběru a analýze dat existuje na trhu spoustu softwaru a hardwaru. Ty zjednodušují výzkumníkům práci, aby vše nemuseli dělat ručně nebo si pamatovat v hlavě. Komerční produkty často umí jak sbírat tak i analyzovat data (Morae [24], Dartfish [6]). Jsou ale i produkty, které nejsou placené, ale nemají tolik možností a taky můžou sloužit jen ke sběru, nebo jen analýze dat. Často vznikají v rámci školních projektů, bakalářských a diplomových prací (IVE [20], UX Tester [26]).

Náplní této práce je vytvoření webové aplikace pro analýzu dat nasbíraných z testů použitelností. Data jsou sbíraná pomocí aplikace UX Tester a ukládána do Firebase databáze [9]. Firebase Realtime Database je NoSQL cloudová databáze. Když se vloží, smažou, nebo upraví data v databáze, tak se změny ihned projeví v aplikacích, které jsou na daná data připojena, takže můžeme v aplikaci provádět analýzu nebo úpravu dat už během testování v reálném čase. Data je taky samozřejmě možné prohlížet a upravovat později. Aplikace bude umět pracovat s videem, fotografiemi a textovými poznámkami.

1.1 Struktura dokumentu

Kapitola 2 popisuje test použitelnosti, jeho průběh, prostředí, typ dat a některé stávající aplikace pro sběr a analýzu dat. Dále obsahuje případy užití, které by měla stávající aplikace

řešit. V kapitole 3 jsem popsal první prototypy, které jsem zkoušel, a které řeší některé základní případy užití. Kapitola 4 se zabývá implementací výsledné aplikace. V kapitole 5 provádím testování aplikace a nakonec kapitola 6 shrnuje výsledky.

Kapitola 2

Analýza

V této kapitole se zabývám testem použitelnosti, popíši o jaký proces se jedná a z jakých kroků se skládá. Máme taky různé druhy prostředí, kde se testuje, nejčastější z nich taky popíši.

Důležitou podkapitolou jsou existující aplikace na sběr dat, tam se budu především zabývat aplikací UX Tester, která slouží ke sbírání dat během testu použitelnosti. Tato práce na ni navazuje, protože cílem je vytvořit webovou aplikaci, která bude zobrazovat, editovat a analyzovat data sbíraná aplikací UX Tester. Dále rozeberu další aplikace, které sbírají a analyzují data a kde by se dalo inspirovat zajímavými prvky.

Nakonec popíši případy užití, které má budoucí aplikace řešit.

2.1 Test použitelnosti

Test použitelnosti (Usability testing) je proces, při kterém jsou participanti pozorování při práci s produktem, který chceme testovat. Uživatelé plní různé smysluplné úkoly a používají produkt stejně, jako kdyby ho používali v reálném životě a prostředí [4].

Participanti jsou reprezentativní vzorek koncových uživatele, na které je produkt cílený. Během testování mají participanti za úkol řešit různé úkoly na prototypu aplikace (nebo v pozdější fázi na reálné aplikaci) stejně tak, jak by je řešili reálně a pozorovatelé je při tom pozorují a dělají si poznámky. Ty jsou poté spolu s video a audio záznamy zpracovány a vyhodnoceny. Potom se můžou udělat úpravy na aplikaci a popř. testovat znova.

Podle toho v jaké části vývoje produktu se nacházíme a jaký je cíl studie, se testování dělí na dva druhy [4]

- *Formativní (formative) testování* - se používá během vývoje produktu, kde je cílem nalezení co nejvíce a nejzávažnějších chyb a jejich opravě. Stačí nám menší počet participantů [18] a jedná se o menší studie, probíhající během vývoje.
- *Souhrnné (summative) testování* - se používá v pozdější části vývoje (např. když je produkt hotový) a testuje se, jestli produkt splňuje všechny požadavky, jestli je použitelný a např. při změnách verze, jestli došlo k zlepšení. Jedná se spíše o kvantitativní testování s více participanty.

2.2 Použitelnost produktu

Testováním si ověřujeme použitelnost našeho produktu. Podle [22] "použitelnost"znamená, že uživatel může provést, to co požaduje a způsobem, kterým očekává, že to bude schopen udělat bez překážek, váhání nebo otázek. Produkt by uživatele během používání neměl frustrovat. Proto by produkt měl splňovat kritéria, jako užitelnost, výkonnost, efektivnost, naučitelnost, spokojenost s používáním a dosažitelnost.

Produkt musí být **užitečný**, jinak ho uživatel nemá důvod používat. Pokud je produkt zbytečný, uživatel ho párkrát použije, ale už se k němu nevrátí.

Výkonnost je rychlost s jakou uživatel dosáhne cíle pomocí produktu. Dá se měřit a chceme, aby náš produkt splnil úkol co nejrychleji.

Efektivní je takový produkt, který se chová tak, jak uživatel očekává, který je ideálně bez chyb a kompletně a přesně splní úkol.

Dosažitelností se rozumí, že uživatel je schopen dosáhnout všech funkcí aplikace/produktu. Produkt by měl být užitečný i když je uživatel dočasně indisponován (např. dočasně zraněn), nebo když je nepříznivé okolí (málo světla apod.).

Naučitelností se rozumí, schopnost uživatele se produkt rychle naučit používat, nebo si rychle vzpomenou na ovládání po delší nečinností. Pokud se uživatel bude muset produkt dlouho učit používat, znamená to, že je složitý a uživatel nemusí mít nervy na to, aby produkt používal.

Uživatel samozřejmě musí být **spokojený** s používáním našeho produktu, pak ho totiž bude chtít používat častěji a taky ho může doporučit dalším potenciálním uživatelům a nebude chtít přejít ke konkurenci.

2.3 Průběh testu

Test se dělí na několik částí, ne všechny jsou povinné a můžou se přeskočit, záleží na typu testu a co od něj požadujeme. Následující kroky si musíme předem naplánovat a podle nich provádět testování [22].

2.3.1 Účel a cíle testu

Obecně popíšeme důvod a motivaci, proč chceme vykonat test použitelnosti, obecně popíšeme problémy, které chceme řešit.

2.3.2 Výzkumné otázky

Jedním z nejdůležitějším bodem jsou správně položeny výzkumné otázky. Ty musí řešit konkrétní problémy, musí být co nejvíce přesné, konkrétní, srozumitelné a musí být měřitelné, aby se na ně dalo jasně odpovědět.

2.3.3 Výběr participantů

Dalším důležitým krokem je výběr participantů. Ti by měli tvořit potenciální koncoví uživatelé, na které je produkt cílený. Podle [22] by minimální počet měl být 10-12 participantů. Na to ale nemusíme mít dostatek prostředků, a proto podle Nielsena [18] nám stačí 5 uživatelů k nalezení 80% chyb, problému nebo nedostatků.

2.3.4 Postup testu

V této části si naplánujeme jak bude test probíhat, jakou použijeme metodologii, promyslíme si práci s participantem (ice-breaking, dotazníky, debriefing apod.) a taky vše načasujeme, abychom dodržovali nějaký plán. Je to důležité, aby pozorovatelé věděli co mají očekávat během testování.

2.3.5 Seznam úkolů

Naplánujeme si seznam úkolů, které bude participant řešit. Bude se jednat převážně o práci s testovaným produktem. K úkolu můžeme taky vymyslet krátký scénář, aby se participant vžil do role.

2.3.6 Testovací prostředí a nástroje

V této části popíšeme v jakém prostředí budeme test provádět, jak ho přizpůsobíme, aby co nejvíce odpovídalo reálným podmínkám. Prostředím chceme co nejvíce vtáhnout participanta do role koncového uživatele.

Popíšeme taky nástroje, který bude participant používat během testování, např. mobilní telefon, počítač, software apod. Nepopisujeme např. kamery, které slouží k monitorování chování participanta.

2.3.7 Role moderátora

Musíme objasnit co bude dělat moderátor testu, jakou roli během testování hraje a kdy bude zasahovat do průběhu testu. Je to důležité proto, aby svým chováním nemátl pozorovatele, aby věděli co od něj můžou očekávat.

2.3.8 Seznam sbíraných dat

V této části musíme objasnit jaká data chceme sbírat. Ta se budou odvíjet od výzkumných otázek. Můžeme sbírat měřitelná data, jako počet chyb, počet správných odpovědí, počet žádostí o pomoc, čas za který participant splnil úkol. Můžeme taky sbírat data, která nejsou měřitelná, jako jsou názory, odpovědi na dotazníky nebo postřehy.

2.3.9 Prezentace výsledků

Zde popíšeme jak bude vypadat naše konečná zpráva, její hlavní sekce.

2.4 Testovací prostředí

Testovat můžeme v různých druzích prostředí. Pokud máme dostatek prostředků, můžeme si postavit laboratoř, kterou si můžeme dobře vybavit, ale participanti se nemusí cítit komfortně, nebo můžeme testovat v terénu. Testováním v terénu se více přiblížíme skutečnému prostředí, ale monitorování participanta je těžší a nemusíme nasbírat tolik dat.

2.4.1 Testování v laboratoři

Laboratoři má velkou výhodu, že si ji můžeme přizpůsobit, tak jak potřebujeme. Dokonce ji můžeme vybavit nábytkem, aby vypadala jako obývací pokoj, nebo jak zrovna potřebujeme, aby se participant lépe cítil. Výhodou je, že do ní můžeme nainstalovat velký počet monitorovacích zařízení, takže můžeme sbírat velký počet dat. Nevýhodou je, že participant se může cítit nesvůj z cizího prostředí a z toho, že je sledován.

Laboratoř se může skládat z více místností [4][22], z jedné, dvou, nebo tří. Nejčastěji se setkáme s dvoumístní laboratoří. Jedna je testovací, která je vybavena kamerami, mikrofony a jinými monitorovacími zařízeními a sedí v ní participant, který testuje produkt, taky s ním může být moderátor, který řídí testování. V druhé místnosti sedí pozorovatelé a sbírají a analyzují naměřená data, může zde taky sedět moderátor a řídit testování pomoci mikrofonu, to má výhodu, že nerozptyluje participanta svou přítomností. Obě místnosti jsou oddělené jednostranně průhledným sklem, aby participanta nerušili pozorovatelé, viz obrázek 2.1.

2.4.2 Testování v terénu

Dalším častým prostředím je testování v terénu [4][22]. V terénu testujeme, když nemáme prostředky na stálou laboratoř, nebo se participant z nějakého důvodu nemůže odstavit k nám do laboratoře, nebo chceme testovat v reálném prostředí, buď venku, nebo doma u uživatele v reálném prostředí.

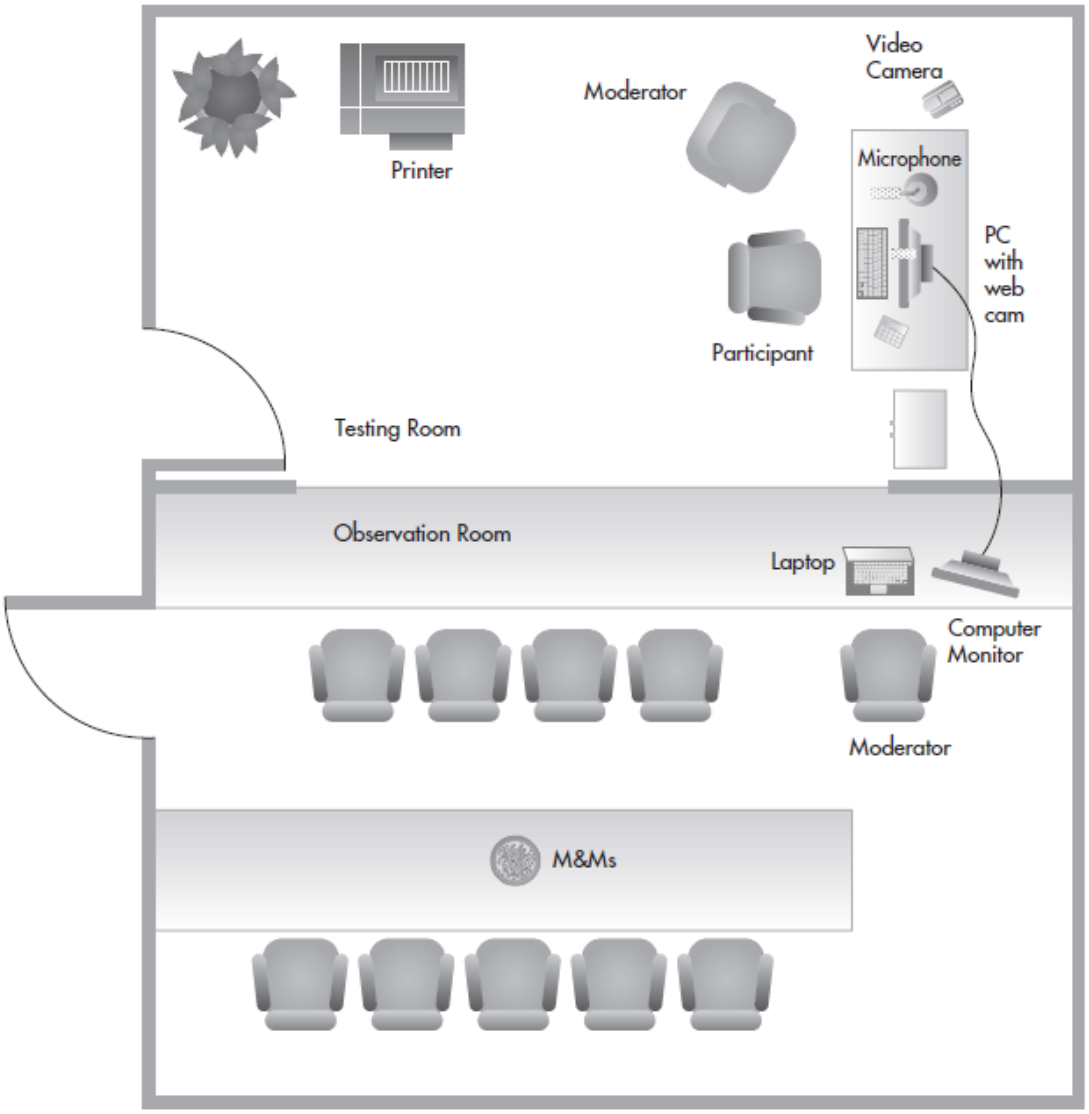
Výhodou testování v terénu je, že simulujeme reálné prostředí, v kterém bude uživatel produkt používat spolu s různými rušivými podměty, které reálně nastanou. Testování je levné, nepotřebujeme tolik vybavení a pohodlné pro participanta (nemusí jezdit za náma).

Hlavní nevýhodou je, že nenasbíráme tolik dat, jelikož nemáme tolik prostředků jako v laboratoři. Testování může být pro nás časově náročné a můžou nastat různé nečekané problémy před nebo během testování, protože nejsme tak připraveni jako v laboratoři a nemůžeme kontrolovat okolí.

2.4.3 Vzdálené testování

Občas není možné testovat s participantem osobně, proto může u určitých druhů produktů (např. software) vzdálené testování [4][22][16]. Stačí nám k tomu telefon nebo počítač, kde se nainstaluje monitorovací software, který bude natáčet obrazovku, nebo zaznamenávat kam uživatel kliknul. Je taky vhodné aby participant myslel nahlas a my si nahrávali jeho poznámky.

Výhodou je, že můžeme testovat participanta, s kterým bychom se jinak nemohli sejít, může se jednat i o zahraničního uživatele. Uživatel dále pracuje ve svém přirozeném prostředí



Obrázek 2.1: Laboratoř z dvěma místnostmi oddělená jednostranně průhledným sklem [22]

a na svém počítači (prohlížeči). Testování je taky rychlejší, snadněji se domluví schůzka s participantem.

Nevýhodou je, že nevidíme participanta, nevidíme jeho řeč těla a jak se chová (to můžeme omezit pomocí videokamery, ale jsou to další starosti navíc). Participant taky nemusí být schopný nainstalovat požadovaný software a obecně může dojít k různým problémům, které nebudeme moct vzdáleně vyřešit. Taky můžou nastát síťové problémy a tím se testování přeruší.

2.5 Sběr dat

Během testování musíme posbírat co nejvíce poznámek a postřehů, je těžké aby to zvládl jeden člověk a při tom řídil test. Proto si musíme pomoct dalšími pozorovateli a technikou. Máme spoustu softwarů, kterými můžeme sbírat různé typy dat. Měli bychom předem vědět jaký typ dat budeme sbírat a jaké výzkumné otázky se budou týkat [22]. Data můžeme dělit na [22]:

- **Data o výkonu (Performance data)** - jsou data, která můžeme měřit, jako čas vykonání úkolu, nebo počet chyb apod. Data můžeme získat během pozorování, nebo ze záznamů.
- **Data o preferencích (Preference data)** - jsou hlavně pocity a názory, které má participant z produktu. Tyto data získáme z dotazníků nebo během debriefingu, nebo např z řeči těla, výrazu tváře apod.

Data můžeme analyzovat kvantitativně a kvalitativně [22]:

- **Kvantitativně** - např. počet chyb, nebo počet negativních názorů.
- **Kvalitativně** - např. závažnost chyby, nebo zdroj chyby, abychom zjistili jak vylepšit náš prototyp. Z negativních názoru můžeme taky dostat nápady, jak nějakou funkci vylepšit.

2.5.1 Typy dat

Data můžeme sbírat různými způsoby, a proto jsou různého typu. Podle zdrojů máme následující typy dat [16]:

- **Video/audio** - jsou nejlepším zdrojem informací. Pokud jsme nestihli pořádně zaznamenat určitý problém, můžeme se k němu vrátit zpětně pomocí video nahrávky. Tu si můžeme taky např. zastavit a prohlédnout si určitý moment do detailů. Z video nahrávky můžeme taky studovat řeč těla nebo výrazy tváře. Pokud máme specializovaný software, můžeme nahrávat dění na obrazovce. Stejně tak můžeme nahrávat audio záznam, toho co participant říká a poznamenat si čas a krátkou poznámku a později se k tomu vrátit a analyzovat odpověď podrobněji.
- **Vlastní poznámky** - je nejjednodušší způsob jak sbírat data. Jelikož během testování není moc času na psaní, tak si pozorovatel zapíše čas a krátkou poznámku, a potom ji synchronizuje spolu s video nebo audio záznamem.
- **Dotazníky a interview** - Před testem dostane participant dotazník, kde můžeme zjistit jeho dřívější zkušenosti s produktem nebo dodatečné informace o participantu. Po testu opět obdrží dotazník, který se bude hlavně týkat proběhlého testu, jaké problém uživatel řešil a jak produkt hodnotí. S participantem si taky o průběhu testu, názorech a postřezích můžeme taky promluvit ve formě interview (to můžeme nahrávat).

- **Data zachycena uživatelským rozhraním nebo použitým zařízením** - Při používání např. počítače nebo mobilního telefonu, můžeme využít softwarů na záznam událostí, které participant vykonal, např. kliknutí na určité tlačítko, pohyb myši, nebo záznam pohybů očí apod. I když samotný monitoring není tak těžký, přesto vyhodnocení nebo nalezení nějakého specifického vzorku chování a z toho odvození závěru nemusí být jednoduché [16].

2.6 Existující aplikace pro sběr a analýzu dat

V této části popíši existující aplikace, které sbírají data (UX Tester [26]) během testu použitelnosti a analyzuje nashromážděná data (IVE [20][16]).

2.6.1 UX Tester

Tato aplikace je diplomovou prací Jakuba Špatného [26]. Jedná se o mobilní aplikaci pro platformu Android. Pozorovatel pomocí aplikace umí získávat data (video, fotografie, poznámky atd.) a odesílat je do realtimové databáze Firebase, nebo si data později prohlížet.

Po přihlášení přes Google, si můžeme založit test a pro ten si předem zadat informační údaje. Můžeme popsat test a vytvořit si seznam participantů a pozorovatelů, kteří budou mít přístup do testu a k datům. To je výhoda, protože více pozorovatelů můžou točit videa, nebo si psát poznámky a všechna data můžou všichni navzájem vidět, takže pro jeden test máme více zdrojů informací. Dále je možnost vytvořit dotazník, který položíme participantovi před testem (pre-test) a dotazník po testu (post-test). Tato část slouží pro správu testu, participantů, dotazníku atd.

Druhou částí jsou tzv. instance testu. Ty náleží jednomu určitému participantovi. Instance slouží jako sběr dat, ukládají se odpovědi uživatele na dotazníky a taky různé druhy souborů a logů, což je nejdůležitější funkcí aplikace.

Během testování můžeme pomocí aplikace rychle zaznamenávat svoje poznatky čtyřmi způsoby. Uložené soubory se uloží a při připojení do sítě se odešlou do databáze.

- **Textový soubor** - Jedná se o jednoduchou textovou poznámku. Není opatřena časovou značkou, takže se nehodí jako doplňující informace např. k videu. Hodí se pro naše vlastní poznámky, nebo popis participanta apod.

- **Log soubor** - Tento typ záznamu je velmi užitečný. V aplikaci se nám otevře okno, kde ve spodní polovině nalezneme tzv. markery. Je jich několik předdefinovaných nebo si můžeme vytvářet svoje další. Tyto markery slouží pro rychlý záznam postřehů, jsou pojmenované např. Bug, Pozitivní názor, Ah-ha moment, Asistence od moderátora apod. Jednotlivé záznamy jsou opatřeny časovou značkou a zobrazují se pod sebou ve vrchní polovině obrazovky. Pokud titulek poznámky nebude dostatečně vysvětlující, můžeme k dané poznámce přidat komentář.

- **Fotografie** - Během testování si můžeme vyfotografovat určitý okamžik a fotografii si uložit a odeslat na server.

- **Video** - Celé testování si většinou pozorovatelé nahrávají, takže v aplikaci nechybí ani tato funkce. Pokud je v testu zapsáno více členů, tak se může točit více videí, např. z jiného úhlu, ve stejném čase. Během natáčení se každých několik sekund vytváří tzv. preview fotografie, které se na pozadí odesílají do databáze, kde je přístupné v reálném čase.

Všechna data se při změně okamžitě uloží do databáze Firebase, která data aktualizuje a pošle událost na všechna zařízení, která na dané události naslouchají, takže se jim data ihned aktualizují.

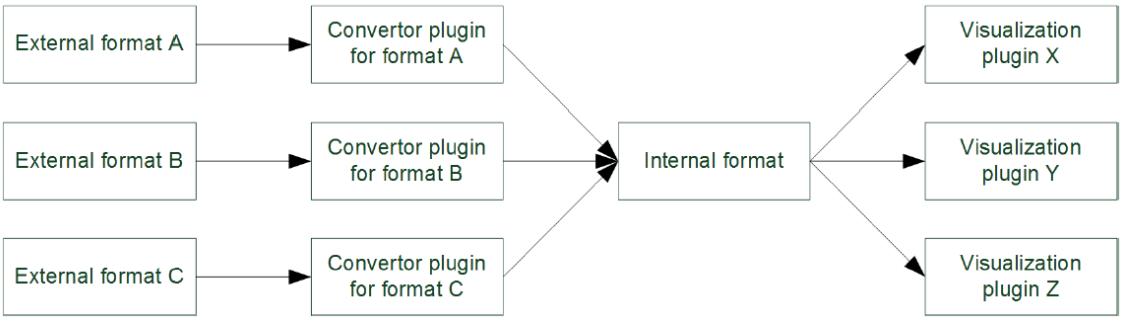
2.6.2 IVE

Integrated Interactive Information Visualization Environment je modulární aplikace na vizualizaci dat z testů použitelnosti. Aplikace vznikla jako diplomová práce Jana Požívila [20] a nyní je vyvíjeno Ing. Ivem Malým, Ph.D. (ČVUT v Praze) [16].

IVE je slouží jako prostředí pro správu projektů a pomocí pluginů dokáže zpracovávat a zobrazovat různé typy dat. Samotné prostředí se vstupními daty neumí pracovat, a proto jsou důležité pluginy třetích stran. IVE podporuje dva druhy pluginů:

- **Pluginy pro konverzi dat (convertor plugins)** - slouží pro konverzi externích vstupních dat do vnitřních struktur, se kterými IVE pracuje.
- **Pluginy pro vizualizaci (visualization plugins)** - slouží pro vizualizaci vnitřních struktur a dat.

Tento systém má velkou výhodu, že pokud jedna firma, která pracuje se svými typy dat, si napíše svůj plugin pro konverzi struktury svých dat na struktury požadované IVE nástrojem, tak potom může využít vizualizační pluginy jiných stran. Vizualizační pluginy tedy nemusí podporovat spoustu druhů formátů dat, ale jenom interní formáty IVE nástroje a tím pádem jsou si pluginy různých výrobců navzájem kompatibilní.



Obrázek 2.2: Model konverze externích dat na interní a vizualizace [20]

Nástroj IVE neslouží jako konvertor dat, ani je nezobrazuje, k tomu slouží pluginy. IVE spravuje projekty a data v interním formátu. Zařizuje komunikaci mezi pluginy a obsahuje kontejnery pro vizualizační pluginy.

2.6.3 Morae

Morae je komerční software pro sběr dat během uživatelského testování a jejich následnou analýzu [24]. Morae se skládá ze tří částí, *Recorder*, který sbírá data, *Observer*, kterým můžeme sledovat data v reálném čase, a *Manager*, který data analyzuje.

Recorder zaznamenává video a audio během uživatelského testování. Taky umí zaznamenávat aktivitu na obrazovce, např. kam participant klikl myší nebo jakou klávesu na klávesnici stiskl.

Observer umožňuje výzkumníkům pozorovat test a data z Recorderu v reálném čase. Výzkumník může také přidávat tzv. Markery, jedná se o jednoduchou poznámku popisující chování participanta. Markery jsou předem definované a slouží jako rychlé poznámky. Dále výzkumník dělá poznámky, jak si participant vede s úkoly (tasks). Jeden záznam pořízeny pomocí Recorderu může sledovat více výzkumníků najednou, to je výhodné, když se testuje ve dvoumístní laboratoři. V jedné místnosti je participant a sbírají se data pomocí Recorderu a v druhé jsou výzkumníci pozorují a dělají si poznámky pomocí Observeru.

Manager slouží k vyhodnocení a analýze dat po skončení testování. V této části může výzkumník také přidávat markery a poznámky. Umí z dat vytvářet grafy anebo je exportovat (např. Excel). Umí také editovat videa, zaznamenaná během testu, pro budoucí prezentaci.

Výhodou je, že se jedná o software, který pokryje celé testování, od sběru dat, tvorbu poznámek až po analýzu dat a export výsledků. Nevýhoda je, že software podporuje jenom operační systém Windows a je komerční a za vysokou cenu (cca. 45000 Kč).

2.6.4 Dartfish

Dartfish je software pro sběr a hlavně analýzu dat, zejména videa [6]. Dartfish poskytuje pokročilé nástroje pro práci s videem a je zaměřen hlavně pro sportovce, kde se zaznamená jejich výkon, a potom analyzuje nebo se porovná s jiným sportovcem.

Díky modulu Dartfish InTheAction je možné s daty pracovat již průběhu testování. Během záznamu se dá tzv. tagovat, tím se vytvoří značka s časem, dobou trvání, názvem, poznámkou aj. Při analýze je potom možnost se ve videu přesouvat pomocí značek, třídit je nebo řadit, což je rychlejší a jednodušší než se jenom přesouvat po časové ose, nebo zrychlovat video.

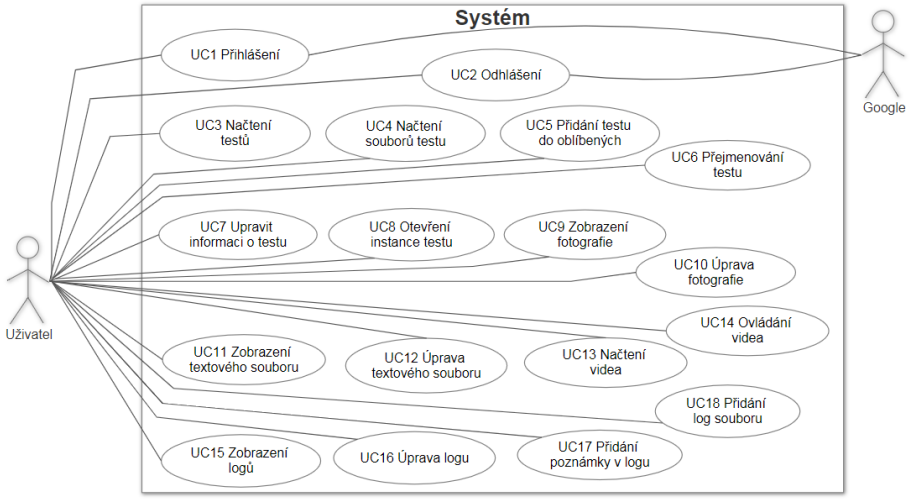
Modul Analyzer obsahuje nástroje sledování a porovnání videi. Modul umí synchronizovaně pouštět více videi vedle sebe anebo dokonce dvě videa přes sebe. Modul taky má funkce jako kreslení do videa (prolínání kreseb a videa) a přidávání popisků, měření vzdáleností, úhlů a časů. Do analýze lze vkládat textové nebo audio komentáře a poznámky.

2.7 Případy užití (Use case)

Před návrhem aplikace jsem sepsal případy užití (viz obrázek 2.3), podle kterých jsem pak navrhoval funkčnost aplikace.

Uživatel si může zobrazit jenom ta data, která vytvořil, nebo ke kterým mu někdo dal práva (uživatel byl přidán jako člen do cizího testu).

Aplikace je propojena s Firebase databází, která pří změně dat ihned pošle událost s daty do aplikace a ta si je aktualizuje, proto musí uživatel být stále připojený k internetu. Firebase zároveň při každém požadavku autorizuje uživatele, proto musí být stále připojený pod svým účtem. Z tohoto důvodu use casy předpokládají, že je uživatel stále přihlášený a připojený k internetu a neřeší chyby, které nastanou, pokud tento předpoklad nebude dodržený. V reálné aplikaci by nedošlo ke změně dat v databázi, nebo by se žádná nestáhla.



Obrázek 2.3: Use case diagram

2.7.1 UC1 Přihlášení

Use case umožňuje se přihlásit přes Google.

Aktéři: Uživatel, Systém, Google

Podmínky pro spuštění: Uživatel je odhlášený.

Základní kroky:

1. Systém zobrazí tlačítko pro přihlášení přes Google
2. Uživatel zmáčkne tlačítko pro přihlášení přes Google
3. Zobrazí se modální okno s výběrem Gmail účtů
4. Uživatel si vybere účet a vyplní heslo
5. Google verifikuje heslo a vrátí systému základní údaje a token
6. Systém přesměruje uživatele na domovskou obrazovku

Alternativní kroky A:

1.1 Pokud je uživatel již v prohlížeči přihlášený do Googlu, systém si od Googlu získá údaje a přesměruje uživatele rovnou na domovskou obrazovku.

Alternativní kroky B:

3.1 Pokud Google nenabídne požadovaný účet, uživatel ho musí zadat ručně, popř. si ho vytvořit

Alternativní kroky C:
5.1 Pokud je heslo špatné, uživatel ho musí zadat znova
Podmínky pro dokončení: Uživatel je přihlášený a nachází se na domovské stránce.

2.7.2 UC2 Odhlášení

Odhlášení přes Google.
Aktéři: Uživatel, Systém, Google
Podmínky pro spuštění: Uživatel je přihlášený.
Základní kroky:
1. Uživatel zmáčkne tlačítko pro odhlášení
2. Systém předá požadavek na odhlášení Googlu
3. Google odhlásí uživatele z aktuálního účtu
4. Systém přesměruje uživatele na přihlašovací obrazovku
Podmínky pro dokončení: Uživatel je odhlášený a nachází se na přihlašovací stránce.

2.7.3 UC3 Načtení testů

Use case umožňuje načíst testy uživatele.
Aktéři: Uživatel, Systém
Podmínky pro spuštění: Uživatel je přihlášený.
Základní kroky:
1. Uživatel si chce prohlédnout svoje testy
2. Systém načte testy z databáze podle ID uživatele
3. Systém zobrazí seznam testů
Alternativní kroky A:
2.1 Pokud pod zadaným ID nejsou data, databáze vrátí prázdný seznam
Alternativní kroky B:
3.1 Pokud seznam neobsahuje data, kontejner pro zobrazení testů bude prázdný
Podmínky pro dokončení: Uživateli se zobrazí jeho testy.

2.7.4 UC4 Načtení souborů testu

Use case umožňuje načíst data náležící testu.
Aktéři: Uživatel, Systém
Podmínky pro spuštění: Uživatel se účastní nějakého testu.
Základní kroky:
1. Systém zobrazí seznam testů, kterých se uživatel účastní
2. Uživatel si vybere požadovaný test
3. Systém stáhne data z databáze

4. Systém zobrazí data daného testu

Alternativní kroky A:

1.1 Pokud je se uživatel neúčastní žádných testů, zobrazí se prázdný seznam

Alternativní kroky B:

4.1 Pokud test neobsahuje některá data, kontejner pro zobrazení požadovaných dat bude prázdný

Podmínky pro dokončení:

Uživateli se zobrazí data požadovaného testu.

2.7.5 UC5 Přidání testu do oblíbených

Přidání testu mezi oblíbené.

Aktéři: Uživatel, Systém

Podmínky pro spuštění: Uživatel má otevřený test.

Základní kroky:

1. Uživatel zmáčkne ikonu hvězdičky pro přidání testu do oblíbených.
2. Systém přidá test mezi oblíbené.

Alternativní kroky:

1.1 Pokud je test již v oblíbených, tak se z oblíbených odebere

Podmínky pro dokončení: Test se zařadí mezi oblíbené.

2.7.6 UC6 Přejmenování testu

Uživatel chce přejmenovat test.

Aktéři: Uživatel, Systém

Podmínky pro spuštění: Uživatel má otevřený test.

Základní kroky:

1. Uživatel si zobrazí možnosti testu a zvolí si editaci názvu testu
2. Systém uloží nový název

Podmínky pro dokončení: Test má nový název.

2.7.7 UC7 Upravit informaci o testu

Uživatel chce upravit informace o testu.

Aktéři: Uživatel, Systém

Podmínky pro spuštění: Uživatel má otevřený test.

Základní kroky:

1. Uživatel si v panelu s informacemi o testu zvolí možnost úpravy.
2. Systém uloží přepíše staré informace za nové.

Podmínky pro dokončení: Informace o testu jsou aktualizované.

2.7.8 UC8 Otevření instance testu

Uživatel si chce načíst některou z instancí testu.

Aktéři: Uživatel, Systém

Podmínky pro spuštění: Uživatel má k dispozici nějaké instance (vytvořil je během testování).

Základní kroky:

1. Uživatel si vybere požadovanou instanci.

2. Systém načte z databáze data instance a zobrazí je.

Alternativní kroky:

2.1. Pokud instance neobsahuje žádná data, zobrazí prázdný seznam.

Podmínky pro dokončení: Uživatel má na obrazovce načtená data požadované instance.

2.7.9 UC9 Zobrazení fotografie

Uživatel si chce zobrazit vybranou fotografii.

Aktéři: Uživatel, Systém

Podmínky pro spuštění: Žádné.

Základní kroky:

1. Uživatel si vybere požadovanou fotografii.

2. Systém zobrazí vybranou fotografii předem určeném panelu/okně.

Alternativní kroky:

2.1. Pokud na serveru už požadovaná fotografie není (byl smazán), systém vypíše příslušné oznámení (aby si uživatel nemyslel, že se fotografie načítá).

Podmínky pro dokončení: Systém zobrazí požadovanou fotografii.

2.7.10 UC10 Úprava fotografie

Uživatel si chce vybranou fotografii zvětšit/zmenšit.

Aktéři: Uživatel, Systém

Podmínky pro spuštění: Systém dokáže zobrazit vybranou fotografii.

Základní kroky:

1. Uživatel pomocí tlačítka + zvětší fotografii.

2. Uživatel pomocí tlačítka - zmenší fotografii.

Podmínky pro dokončení: Systém uloží upravenou fotografii.

2.7.11 UC11 Zobrazení textového souboru

Uživatel si chce zobrazit vybraný textový soubor.

Aktéři: Uživatel, Systém

Podmínky pro spuštění: Žádné.

Základní kroky:

1. Uživatel si vybere požadovaný textový soubor.

2. Systém na příslušném místě (okně/panelu) text zobrazí.

Podmínky pro dokončení: Systém zobrazí požadovaný textový soubor.

2.7.12 UC12 Úprava textového souboru

Uživatel si chce vybraný textový soubor upravit.

Aktéři: Uživatel, Systém

Podmínky pro spuštění: Žádné.

Základní kroky:

1. Uživatel si v panelu s textem zvolí možnost úpravy textu.

2. Systém upravený text uloží.

Alternativní kroky:

2.1. Pokud je text prázdný, vytvoří se nový obsah souboru.

Podmínky pro dokončení: Systém uloží textový soubor.

2.7.13 UC13 Načtení videa

Uživatel si chce zobrazit vybrané video.

Aktéři: Uživatel, Systém

Podmínky pro spuštění: Žádné.

Základní kroky:

1. Uživatel si vybere požadované video.

2. Systém zobrazí vybrané video na předem určeném místě (panelu/okně).

Alternativní kroky:

2.1. Pokud na serveru už požadované video není (bylo smazáno), systém vypíše příslušné oznámení (aby si uživatel nemyslel, že se video načítá).

Podmínky pro dokončení: Systém zobrazí požadované video.

2.7.14 UC14 Ovládání videa

Uživatel chce zobrazované video ovládat.
Aktéři: Uživatel, Systém
Podmínky pro spuštění: Systém přehrává uživateli video.
Základní kroky:
1. Uživatel pomocí ovládací lišty ovládá video (Play, Pause, přetáčení, popř. rychlost přehrávání).
2. Systém podle podle požadovaného pokynu ovládá video.
Podmínky pro dokončení: Uživatel ovládá video podle potřeby a možnosti systému.

2.7.15 UC15 Zobrazení logů

Uživatel si chce zobrazit poznámky v požadovaném log souboru.
Aktéři: Uživatel, Systém
Podmínky pro spuštění: Žádné.
Základní kroky:
1. Uživatel si vybere požadovaný log soubor.
2. Systém stáhne z databáze poznámky uložené v log souboru a zobrazí je jako seznam.
Alternativní kroky:
2.1. Pokud je log soubor prázdný, systém na to upozorní uživatele (např. "Žádná data").
Podmínky pro dokončení: Systém zobrazí požadované poznámky do seznamu.

2.7.16 UC16 Úprava logu

Uživatel chce změnit poznámku v logu.
Aktéři: Uživatel, Systém
Podmínky pro spuštění: Log soubor obsahuje aspoň jednu poznámku.
Základní kroky:
1. Uživatel si vybere poznámku, kterou chce upravit a změní její typ nebo komentář.
2. Systém upravenou poznámku uloží (časová značka zůstane původní).
Podmínky pro dokončení: Systém uloží upravenou poznámku logu.

2.7.17 UC17 Přidání poznámky v logu

Uživatel chce přidat poznámku v logu.
Aktéři: Uživatel, Systém
Podmínky pro spuštění: Log soubor, do kterého chce uživatel přidat poznámku, je již vytvořený.
Základní kroky:
1. Uživatel vybere typ poznámky a připojí k němu komentář (nepovinné) a uloží poznámku.
2. Systém vytvoří poznámku a připojí k ní aktuální časovou značku
3. Systém uloží novou poznámku do logu a databáze.
Podmínky pro dokončení: Systém uloží novou poznámku do logu.

2.7.18 UC18 Přidání log souboru

Uživatel chce přidat nový log soubor.
Aktéři: Uživatel, Systém
Podmínky pro spuštění: Není žádný jiný vytvořený log soubor.
Základní kroky:
1. Uživatel vybere možnost pro vytvoření logu.
2. Systém vytvoří nový log soubor.
3. Systém uloží log soubor do databáze.
Podmínky pro dokončení: Systém vytvoří nový prázdný log soubor.

Kapitola 3

Navrh řešení

Vytvořil jsem 2 prototypy, kde řeším některé důležité případy užití (use case). Tyto prototypy neřeší všechny případy užití, ale jenom ty, které ověřují stěžejní funkce, hlavně připojení k databázi Firebase (přihlášení, odhlášení) a načítání a zobrazovaní dat z ní, jako jsou text, logy, fotografie a video. Ostatní případy užití ale už kompletní aplikace řeší. Jedná se o jednoduché prototypy, abych zjistil jestli jsou vhodné pro daná data a danou aplikaci.

3.1 První prototyp

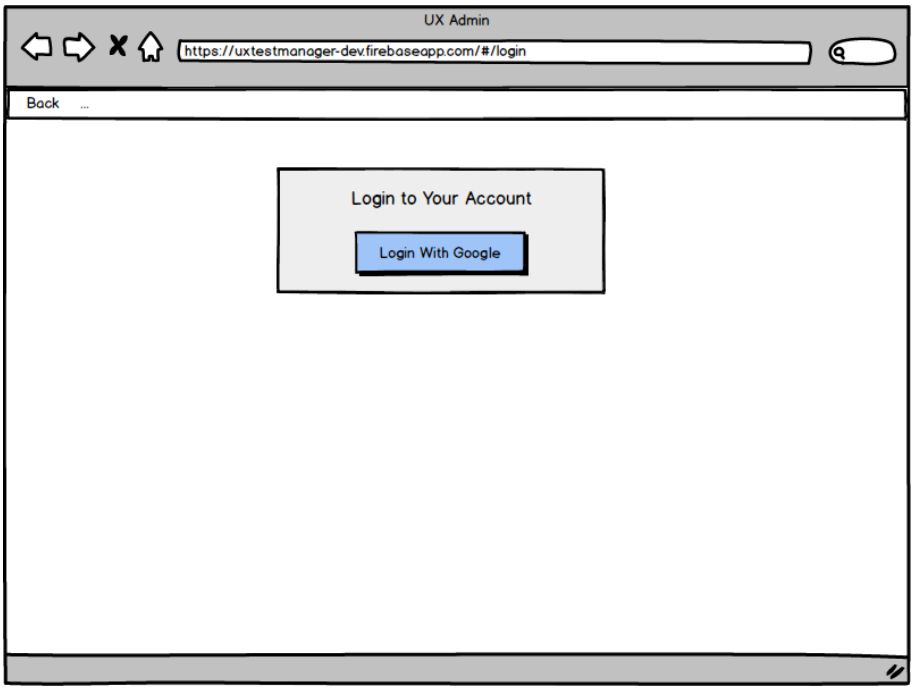
První prototyp slouží jako jednoduchá aplikace, ve které jsem vyzkoušel načítání a prohlížení dat z databáze Firebase. Prototypem uživatel může procházet soubory testu, nebo si otevřít konkrétní instanci a prohlédnout si informace o ní a soubory, které jsou v ní uložené, popř. je stáhnout. Data nelze měnit nebo upravovat. Prototyp řeší jenom základní use casy, jako je přihlášení do databáze, načítání různých dat, zvlášť mediálních souborů a logů, data lze také stáhnout.

3.1.1 UC1 Přihlášení

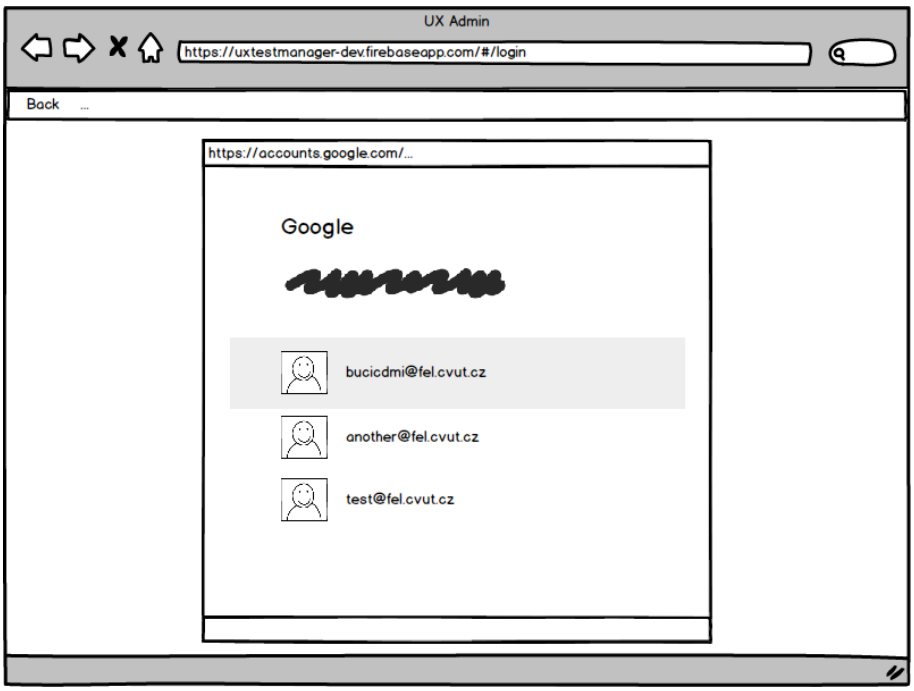
Obrazovka se skládá z hlavičky, která je na všech obrazovkách. Na ní se nachází tlačítko *Back*, které uživatele naviguje v historii o jeden krok zpět. Napravo je tlačítko *Logout/Login*, podle toho, jestli je uživatel přihlášený/odhlášený.

Uprostřed obrazovky (Obrázek 3.1) je tlačítko, pro přihlášení přes Google. Po jeho kliknutí se zobrazí modální okno s výběrem účtu a zadání hesla.

V modálním oknu (Obrázek 3.2) vyberu účet a zadám heslo, pokud je správné, tak se přihlásím a aplikace mě přesměruje na domovskou obrazovku.



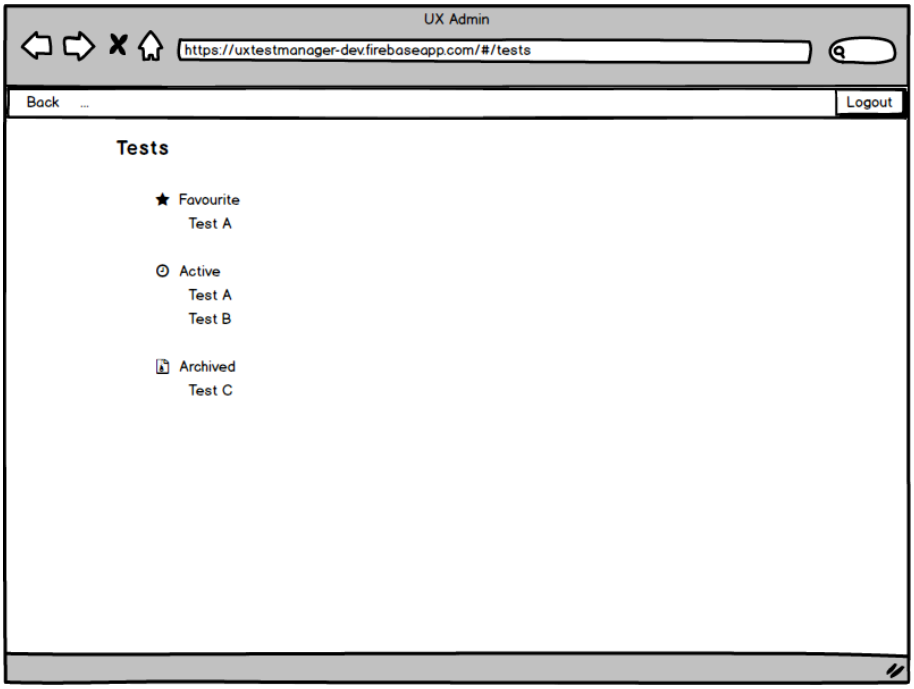
Obrázek 3.1: Přihlášení do Google



Obrázek 3.2: Výběr účtu

3.1.2 UC3 Načtení testů

Na domovské obrazovce máme seznam testů (obrázek 3.3), rozdělený podle toho, jestli je v *Oblíbených, Aktivní,* nebo *Archivovaný*. Pokud je jeden test ve více skupinách, zobrazí se všude. Po vybrání testu, přejdu na jeho detailní popis.



Obrázek 3.3: Seznam testů

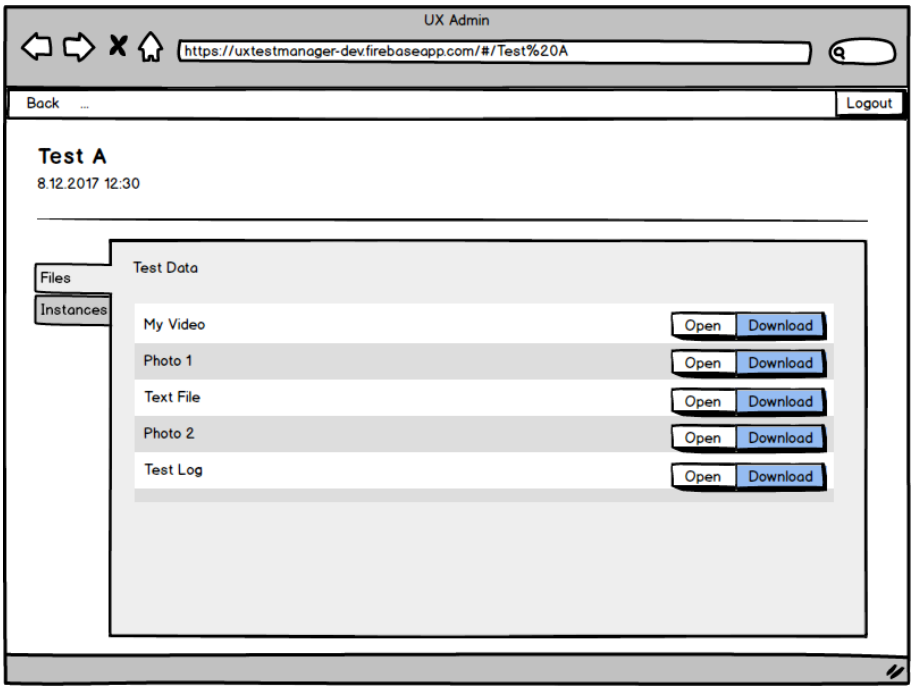
3.1.3 UC4 Načtení souborů testu

Ve vrchní části je název testu a datum. Pokud test obsahuje nějaké soubory anebo instance, tak se zobrazí tlačítka, pro jejich zobrazení. Pokud kliknu na tlačítko *Test Files*, zobrazí se mi seznam souborů (obrázek 3.4), které jsou uložené pro daný test, tlačítkem *Instances* si vyberu požadovanou instanci a aplikace mě přesměruje na její detail.

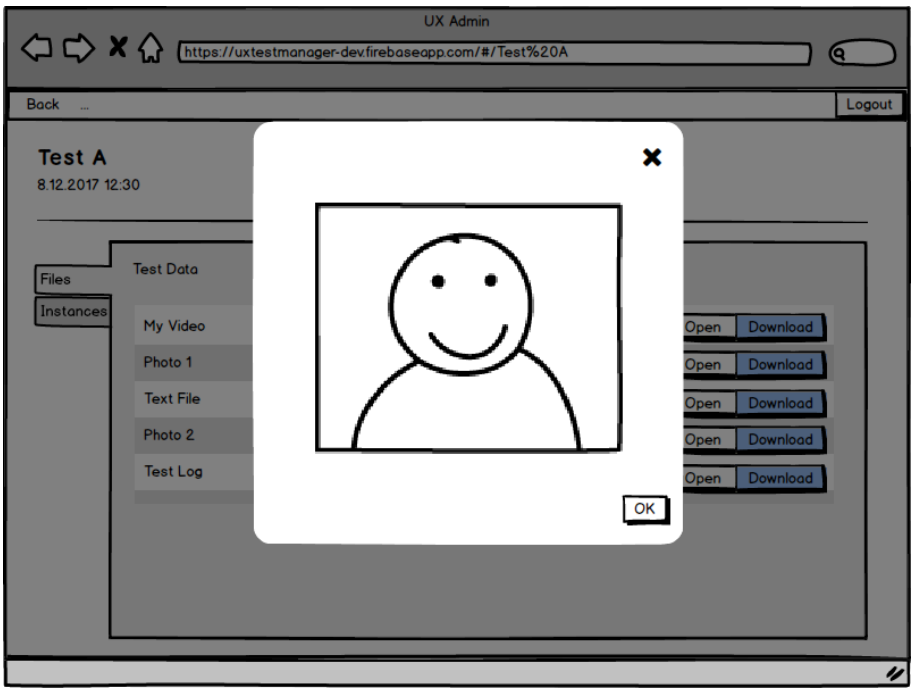
Aplikace načte z databáze Firebase data, která jsou uložená pro daný test. Soubory se zobrazjují jako seznam. Může se jednat o log soubor, textový soubor, fotografii nebo video. Vlevo je název a typ souboru, vpravo jsou tlačítka *Open* a *Download*, pro zobrazení souboru a dat, nebo pro jejich stažení.

3.1.4 UC9 Zobrazení fotografie

Pokud si uživatel chce zobrazit fotografii, otevře se modální okno, fotografie se načte z databáze a zobrazí (obrázek 3.5). Tlačítkem *OK*, kliknutím na křížek anebo mimo okno, se okno zavře.



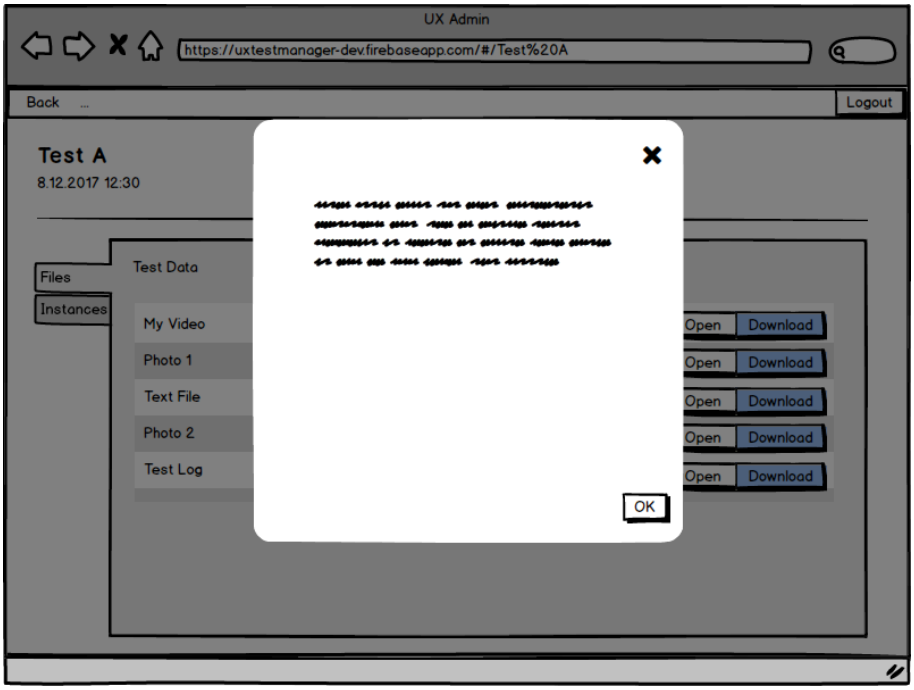
Obrázek 3.4: Soubory testu



Obrázek 3.5: Zobrazení fotografie

3.1.5 UC11 Zobrazení textového souboru

Pokud si uživatel chce zobrazit textový soubor, otevře se modální okno, text se načte z databáze a zobrazí (obrázek 3.6). Tlačítkem *OK*, kliknutím na křížek anebo mimo okno, se okno zavře.



Obrázek 3.6: Zobrazení textového souboru

3.1.6 UC13 Načtení videa

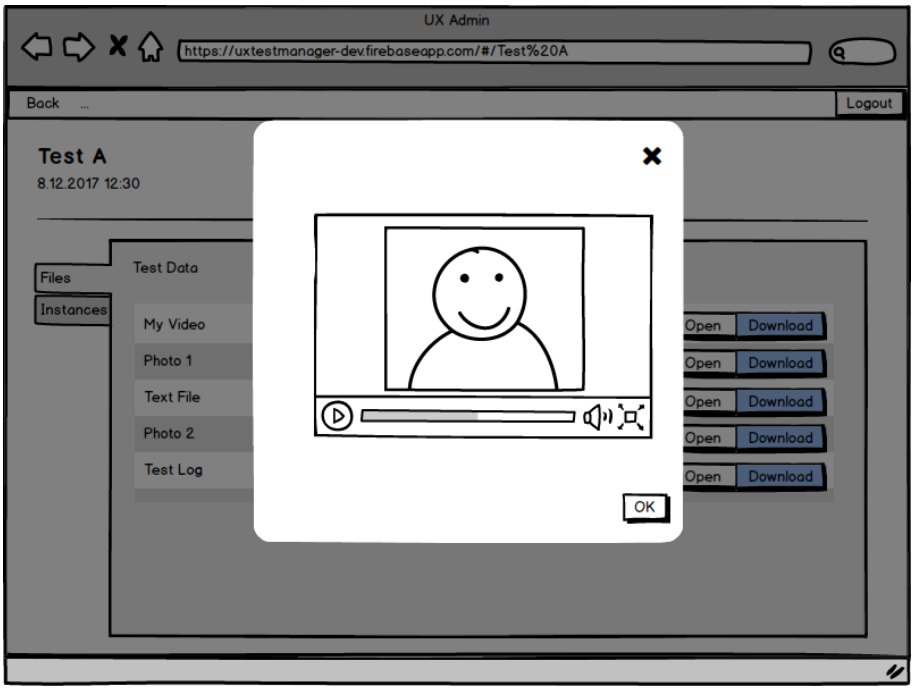
Pokud si uživatel chce zobrazit video, otevře se modální okno, video se načte z databáze a zobrazí (obrázek 3.7). Tlačítkem *OK*, kliknutím na křížek anebo mimo okno, se okno zavře.

3.1.7 UC15 Zobrazení logů

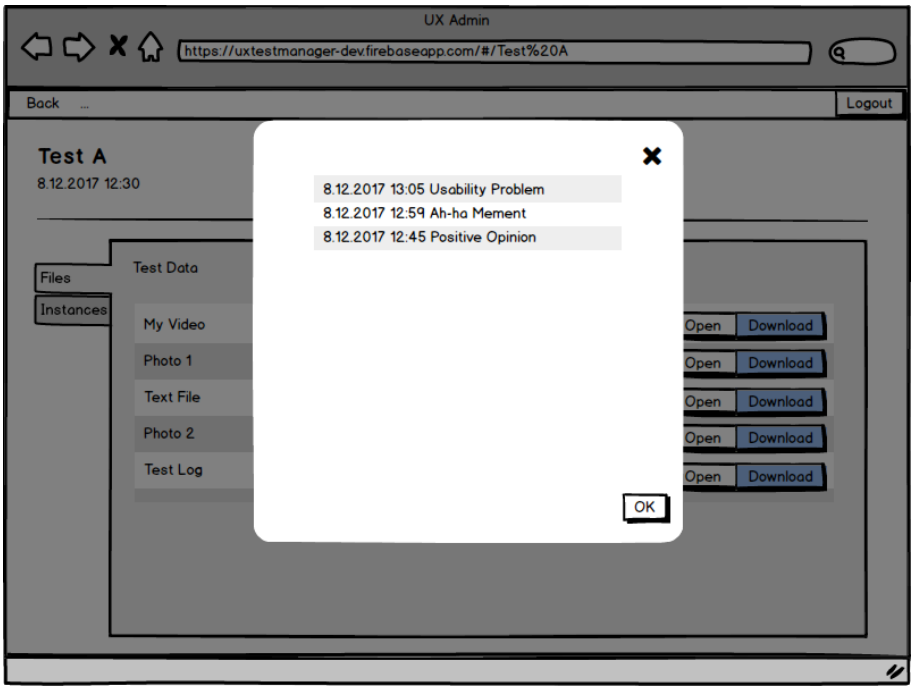
Pokud si uživatel chce zobrazit logy, otevře se modální okno, logy se načte z databáze a zobrazí se seznam logů (obrázek 3.8). Vlevo je čas, kdy byl log pořízený, vpravo je název logu a komentář, pokud byl zadán. Tlačítkem *OK*, kliknutím na křížek anebo mimo okno, se okno zavře.

3.1.8 UC8 Otevření instance testu

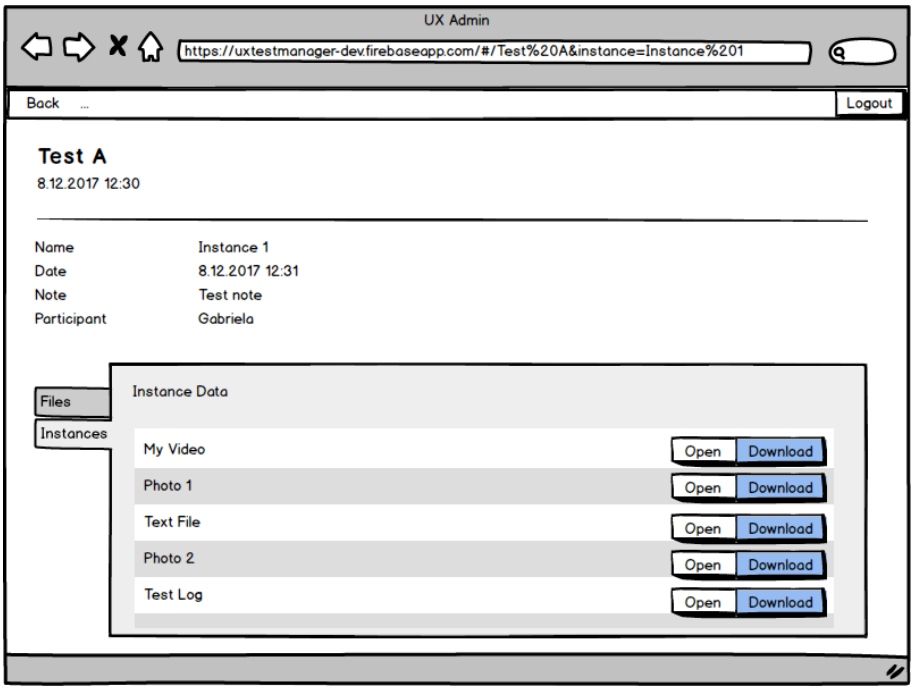
Aplikace načte z databáze požadovanou instanci a zobrazí její dat, jako je jméno datum vytvoření, poznámky a participanty. Zobrazí taky seznam souborů, které instance obsahuje (obrázek 3.9), stejně jako je to u testu.



Obrázek 3.7: Načtení videa



Obrázek 3.8: Zobrazení logů



Obrázek 3.9: Instance testu

3.1.9 Závěr

První prototyp dokázal jednoduše procházet soubory, prohlížet si je a popř. si je stahovat, přesto to ale není vhodné řešení pro naši aplikaci. Uživatel se musí proklikávat spousty okny, než si zobrazí data, může si zobrazit jen jeden soubor najednou apod. Špatně by se mu tedy data analyzovala, když by je neviděl v kontextu s ostatními. Proto jsem navrhl druhý prototyp, který je více podobný vývojovým prostředím (IDE).

3.2 Druhý prototyp

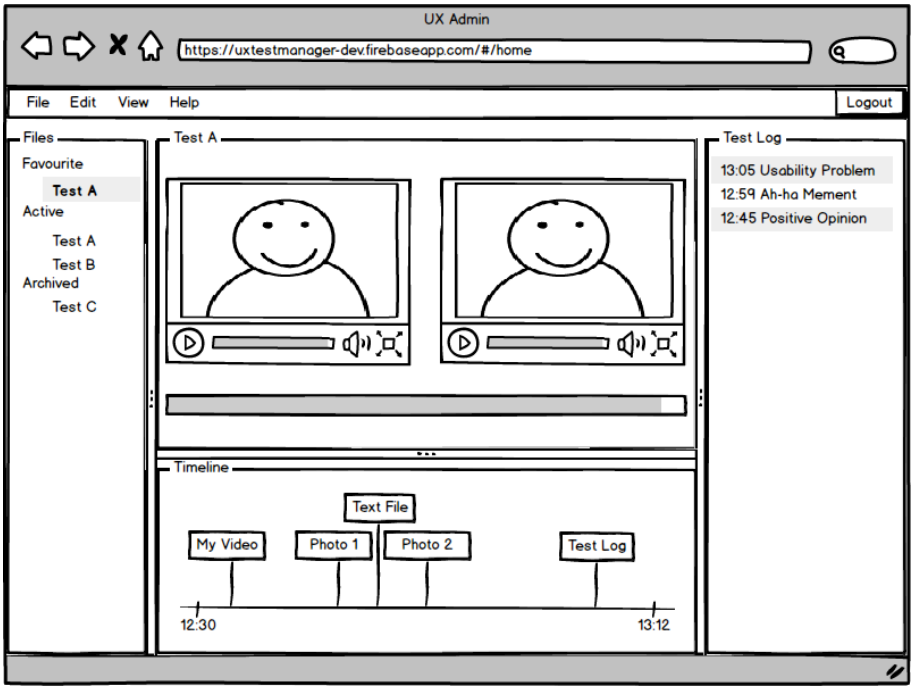
Druhý prototyp je inspirovaný mnohými aplikacemi (např. Integrated Visualization Environment), kde se okno skládá z několika panelů, v levém panelu je strom filesystému a na dalších panelech jsou zobrazována data. Po přihlášení (to je implementované stejně jako v prvním prototypu) se uživateli zobrazí jedno okno, ve kterém se nacházejí kontejnery pro zobrazení různých typů dat. Tím je docíleno, že se všechna potřebná data zobrazují uživateli najednou a má o nich přehled.

Hlavním výstupem testu použitelnosti je video, které zachycuje participanta při testování. Hlavní část obrazovky tedy bude tvořit kontejner pro video, kdyby test obsahoval další videa (např. pohled z jiného úhlu) bude se moct další video přidat.

Dalším důležitým prvkem je časová osa, na které by byly zobrazeny značky pro všechny logy, pořízené fotografie, nebo videa. Jednalo by se o přehled a po kliknutí na danou značku, by se obsah zobrazil v panelu vpravo.

3.2.1 Otevření testu a zobrazení dat

Okno se skládá z několika panelů, vlevo je seznam testů uživatele, rozdělené podle toho jestli patří mezi *Oblíbené*, *Aktivní* nebo *Archivované*. Po vybrání testu se na prostřední části obrazovky načtou jeho data. Vrchní panel obsahuje videa, která jsou důležitým zdrojem informací testu použitelnosti. Ve spodní části je časová osa, na které jsou zobrazeny události, např. pořízení fotografie, nebo záznam do logu apod. Pokud uživatel vybere některou událost, její obsah se zobrazí v panelu vpravo, na obrázku můžeme vidět seznam logů. Nahoře se nachází ovládací lišta a tlačítko *Logout* pro odhlášení uživatele (viz obrázek 3.10).



Obrázek 3.10: Návrh druhého prototypu

3.2.2 Závěr

Tento layout je přehlednější. Výhodou je taky zobrazení více panelu s různými daty v jednom okně. Uživatel má o datech přehled a lépe vidí mezi nimi souvislosti. Proto jsem se ho rozhodl implementovat tento prototyp.

Kapitola 4

Implementace

V této kapitole se budu zabývat implementací aplikace. Budu vycházet z návrhu druhého prototypu, protože jeho layout je podobný jiným aplikacím, je srozumitelnější a uživatel má přehled o různých datech na jednom místě (viz Obrázek 3.10).

Aplikace je psaná pomocí webového frameworku Angular 5 [1], data jsou uložena v real-time databázi Firebase a jsou s aplikací synchronizována, jejich změna se v aplikaci projeví v reálném čase.

Aplikace je k dosažení na url adrese https://uxtestmanager-dev.firebaseapp.com/.

4.1 Angular

Angular je MVC (Model-View-Controller) framework, který rozšiřuje HTML o různé elementy a atributy nebo vytváří vlastní, obsahuje Two way data-binding [2], odděluje zobrazovací logiku (view) od aplikační (model) aj. V aplikaci se používají *komponenty*, které se starají o aplikační logiku a jsou propojeny se šablonou (template), která se stará o zobrazovací logiku. Dále se využívají tzv. *services*, díky kterým můžou komponenty mezi sebou komunikovat, nebo obsahují sdílené funkce, které můžu volat z různých komponent.

Aplikace používá Angular verzi 5.2.0, což je důležité kvůli kompatibilitě s jinými knihovnami, které aplikace využívá.

4.1.1 Struktura projektu

Projekt v Angularu obsahuje adresáře obsahující zdrojové kódy, psané v jazyce TypeScript a šablony v HTML, adresáře s javascriptovými knihovnami, adresář s testy a několika konfiguračními soubory. Níže popíši typickou strukturu.

Adresáře:

- **/src/app** - vkládají se sem komponenty (viz 4.1.2), každá je v samostatném adresáři a skládá se *.css (styly), *.html (html šablona), *.spec.ts (testování) a *.ts (controller) souborů. Všechny controllery, moduly, service aj. musí být importované v /src/app/app.module.ts.

- **/src/assets** - zde můžeme ukládat obrázky, ikony, videa, jiné *.css soubory nebo javascriptové knihovny.
- **/src/model** - zde jsou uloženy modely, které můžeme používat jako typ u proměnné, např: private user: User = new User().
- **/src/services** - zde se uloží service (viz 4.1.4), které obsahují sdílené proměnné nebo funkce.
- **/node_modules** - v tomto adresáři jsou uloženy všechny knihovny které jsou specifikované v *package.json*, a na kterých jsou závislé.
- **/e2e** - adresář slouží k testování aplikace.
- **/dist** - tento adresář vznikne při vytvoření buildu aplikace, tento adresář se nahraje na server.

Konfigurační soubory:

- **/.angular-cli.json** - obsahuje nastavení projektu, hlavně názvy základních adresářů a souborů a cesty k ním.
- **/package.json** - jsou zde zapsány moduly (knihovny), které má npm[1] stáhnout, nebo názvy scriptů, které přes npm můžeme spouštět.
- **/protractor.conf.js** a **karma.conf.js** - slouží pro nastavení testování Angularu.
- **/firebase.json** a **/.firebaserc** - konfigurační soubory potřebné pro hosting naší aplikace na službě Firebase Hosting [11].

4.1.2 Komponenta

Angular komponentu můžeme chápat jako html element, kterému můžeme definovat svoje atributy (direktivy) a můžeme ho používat vícekrát kdekoliv v aplikaci. Komponenta obsahuje tři druhy dat.

- **Šablonu (.html)** - vytváří vzhled elementu.
- **Třída (.ts)** - obsahuje funkce a proměnné, které se můžou použít v šabloně.
- **Styly (.css)** - jsou používané pouze danou komponentou.

V Angularu se jednotlivé stránky tvoří jako komponenty, a mezi nimi se přechází pomocí modulu *Router*.

Public proměnné a funkce se můžou používat i v šabloně. Díky tzv. data bindingu (viz podkapitola 4.1.3) se změna v hodnotě proměnné projeví hned v šabloně a naopak, přitom uživatel nemusí aktualizovat stránku, což má výhodu z hlediska dynamičnosti stránky a kratšího a jednoduššího kódu.

[1] Node.js Package Manager - balíčkový manager prostředí Node.js

4.1.3 Two way data binding

Jedná se o synchronizaci dat mezi modelem (controller) a view (šablona). Pokud vytvořím public proměnou v komponentě a použiji ji v šabloně, tak budou synchronizované a pokud změním její hodnotu v komponentě, okamžitě se změní i v šabloně a naopak. K tomu mi slouží buď dvojité složené závorky (jen pro čtení) anebo direktiva *ngModel*. Hranaté závorky značí posílání dat z šablony do controlleru, kulaté závorky naopak, obě závorky značí obousměrnou komunikaci - proto "two way data binding".

```
<div>Jméno: {{user.name}}<\div>
<input type="text" [(ngModel)]="user.name">
```

4.1.4 Service

Service je třída, která obsahuje public funkce a proměnné, které se můžou volat z jiných komponent a service, takže ty funkce se nemusí programovat vícekrát. Angular obsahuje také spoustu vestavěných service, např. Http nebo Router.

Přes public proměnné si spolu můžou komponenty předávat data (např. informace o přihlášeném uživateli). Jelikož v této aplikaci budu mít jednu obrazovku a v ní bude více komponent, budu potřebovat service, aby si předávali data a informace. K tomu taky slouží tzv. *Subject*.

Subject je třída, pomocí které jedná komponenta pošle data druhé, která je očekává, ale neví, kdy budou přítomny. Např. při kliknutí na položku v časové ose, chci data o položce poslat do určité komponenty a tam zobrazit detaily o položce. Service bude vypadat následovně:

```
private subject = new Subject<any>();

constructor() {}

setData(data: any) {
  this.subject.next(data);
}

getData(): Observable<any> {
  return this.subject.asObservable();
}
```

Pomocí metody *this.subject.next(data)* se data odešlou všem, kdo o ně zažádá přes metodu *this.subject.asObservable()*. Data z *Observable<any>* se získají pomocí metody *Observable<any>.subscribe()*

```
this.subscription = this.alertSvc.getData().subscribe((data: string) => {
  this.inner.nativeElement.innerHTML = data;
});
```

4.1.5 Získávání dat z databáze Firebase

Práce s realtime databází Firebase je odlišná od jiných databází, kde si aplikace požádá o data a pak je zobrazí a pokud se data změní, tak aplikace zobrazuje stará data do té doby, než si zase nezažádá o nová data, např. při aktualizaci stránky. To s použitím Firebase odpadá.

Získávání dat z databáze zprostředkovává modul *AngularFireDatabaseModule*, ten obsahuje service *AngularFireDatabase*, která má funkce *object(path) a list(path)*. Pomocí nich a funkce *valueChanges()* se získávají data z databáze (objekt nebo pole objektů). Funkce *valueChanges()* vrací *Observable<T>* a každý, kdo si ho podepíše (např. *Observable<T>.subscribe(data => zobrazData(data))*), tak získá nová data a může si aktualizovat hodnoty v šabloně a zobrazit nejnovější data. Funkce *subscribe* vrací třídu *Subscription*, kterou musíme zničit potom, co ji přestaneme potřebovat, jinak při aktualizaci stránky by se vytvářeli pořád nové a zabírali by místo v paměti. To jsem vyřešil tak, že všechny *Subscription*, které jsem vytvořil v aktuální komponentě vložím do pole a při opuštění komponenty (např. při aktualizaci stránky), tak je přes for-cyklus zničím metodou *unsubscribe()*.

Příklad použití modulu:

```
const subcription = this.af.object(path)
    .valueChanges()
    .subscribe((data: any) => {
        this.aktualizuj(data);
    });
this.subscriptions.push(subcription);
```

Vždy, když se změní data, která jsou na cestě *path*, tak se zavolá funkce *this.aktualizuj(data)*.

4.1.6 Použité knihovny

V Angularu se často využívá externích JS knihoven a pro jejich správu slouží balíčkový manager *npm (Node.js package manager)*. V konfiguračním souboru *package.json* se vypisují všechny knihovny, které projekt chce využívat a pomocí příkazu *npm install* se tyto knihovny (a knihovny, na kterých jsou závislé) nainstalují do adresáře */node_ modules*. Knihovny se musí také naimportovat v souboru */src/app/app.module.ts*.

Výhodou je, že na internetu [19] je už spousta knihoven, které řeší nejrůznější problémy, takže je můžeme používat a nemusíme je implementovat.

Nevýhodou je, že při aktualizaci knihoven pomocí příkazu *npm install* se všechny knihovny smažou, takže pokud bychom je upravovali, tak přijdeme o změny. V tom případě si můžeme knihovnu uložit zvlášť, např. ve složce */src/assets*.

Tady jsou některé důležité knihovny, které jsem v aplikaci použil:

- **Vis.js[5]** - jedná se o vizualizační knihovnu, mimo jiné, k zobrazení dat v časové ose - timeline. Časovou osu jsem využil k zobrazení souborů v čase. Jednotlivé typy souborů se liší barevně. Aktuální čas je zobrazen modrou vertikální čarou, která se s časem posouvá. Knihovna umožňuje spoustu možností, jak pracovat s předměty obsaženy v časové ose.

- **Angularfire2[7]** - je knihovna pro komunikaci mezi Angularem a Firebase (viz podkapitola 4.2). Obsahuje funkce pro přihlášení do Firebase, pro načítání a ukládání dat do realtime databáze, pro načítání a ukládání souborů do Storage aj. Pokud chceme používat nejnovější verzi 5.0.0, musíme mít také Angular verzi 5.
- **Golden Layout[8]** - je knihovna pro správu oken ve webové aplikaci. V této aplikaci je použita pro zobrazení více komponent na jedné stránce, aby uživatel měl k dispozici více dat. Okna se dají také přemísťovat a měnit jejich velikost.

4.2 Firebase

Firebase je platforma spravovaná společností Google. Nabízí NoSQL databázový systém s širokým výběrem služeb (např. úložiště, hosting aj. viz [13]). V této aplikaci jsou využity následující služby.

- **Firebase Auth** - slouží pro přihlášení do databáze.
- **Firebase Realtime Database** - NoSQL databáze poskytující synchronizaci dat mezi klienty v reálném čase.
- **Firebase Storage** - jedná se o úložiště souboru jako jsou obrázky, audio nebo video.
- **Firebase Hosting** - Firebase nabízí i web hosting, kde jsou uloženy CSS, HTML a JavaScript soubory.

4.2.1 Firebase Realtime Database

Firebase Realtime Database je cloudová NoSQL databáze, data nejsou uložena v tabulkách, ale jsou uchovávána jako JSON. Data jsou mezi klienty synchronizovaná, pokud se změní, všichni obdrží nejnovější data v reálném čase.

Příklad struktury databáze:

```
{
  users: {
    testUser: {
        email: "test@gmail.com",
        name: "Test User",
        messages: {m1: "...", m2: "...", ...}
    },
    admin: {
        email: "admin@gmail.com",
        name: "Admin",
        messages: {m1: "...", m2: "...", ...}
    }
  }
}
```

Další výhodou je, že k datům se může přistupovat rovnou z aplikace, takže není potřeba serverové části. K získání dat se používá URL syntaxe např:

```
https://app-name.firebaseio.com/users/testUser
```

K datům může přistupovat kdokoliv, proto je doporučené použít bezpečnostní pravidla, tzv. Firebase Realtime Database Security Rules, kde se definuje k jakým datum má jaký uživatel práva a jaká data se můžou ukládat [10].

4.2.1.1 Struktura databáze

Data jsou uložena jako JSON objekty a tedy cela databáze má stromovou strukturu. Data se ukládají jako klíč: hodnota. Klíč si může uživatel zvolit sám, nebo se pomocí funkce *push()* vygeneruje sám (knihovna *Angularfire2* ještě nabízí funkci *createPushId()*, kdy se vygeneruje ID, ve stejném formátu jako z funkce *push()*, a uživatel s ním může dál pracovat). Hodnota se může ukládat jako jeden záznam nebo celý objekt [15].

Při žádosti o data databáze vrátí uzel a všechny jeho potomky, což může být problém, pokud jsou data do sebe hodně vnořená a rozvětvená. Může docházet k přenosu zbytečně mnoha dat, pokud potomky nepotřebuje. To je nevýhoda hlavně u mobilních aplikaci z omezeným připojením. Proto Firebase doporučuje nevnořovat data, přestože databáze povoluje 32 úrovní. Někdy je výhodné data rozdělit anebo i zduplikovat (tzv. denormalizace), aby se omezilo vnoření. Např. předešlý příklad se strukturou databáze, pole *messages* se může vložit pod jiný uzel. Když si budu chtít získat email uživatele, tak nebudu muset stahovat všechny jeho zprávy [15].

```
{
  users: {
    testUser: {
        email: "test@gmail.com",
        name: "Test User"
    },
    admin: {
        email: "admin@gmail.com",
        name: "Admin"
    }
  },
  messages: {
     testUser: {m1: "...", m2: "...", ...},
     admin: {m1: "...", m2: "...", ...}
  }
}
```

Jelikož moje aplikace pracuje s projekty vzniklými v aplikaci UX Tester, tak jsem databázi, spolu s daty a strukturou, měl k dispozici [26].

4.2.1.2 Bezpečnostní pravidla

Firebase Realtime Database Rules poskytuje prostředek pro zabezpečení dat, kontroluje kdo má k jakým datum práva, validuje jestli vložená data mají správný formát nebo nastavuje indexování.

Základní kontrola je, aby byl uživatel přihlášený, bez toho se k datum uživatel nedostane. Potom se každému uzlu v databázi dá nastavit čtyři druhy pravidel a před každou operací se nejdříve zkontroluje, jestli požadavek splňuje daná pravidla.

- **.read** - pokud je pravidlo splněné, uživatel může číst data.
- **.write** - pokud je pravidlo splněné, uživatel může upravovat nebo vkládat data.
- **.validate** - kontrola formátu zapisované hodnoty (např. délka řetězce), typu nebo jestli objekt obsahuje určité atributy.
- **.indexOn** - nastavuje, podle kterých potomků se má indexovat (kvůli řazení).

Firebase Database Rules obsahuje proměnné, které se používají v pravidlech, např. *auth* jsou data o přihlášeném uživateli získaný pomocí Firebase Authentication. Z proměnné *auth* získáme UID přihlášeného uživatele a to zkontrolovat se záznamem v databázi, např. aby data mohl měnit jen vlastník [14].

```
{
  users: {
    xnuf7af8nfi2fbFfw211: {email: "...", name: "..."},
    dbjdb6dtw7DWD78wdDV7: {email: "...", name: "..."},
    aodn878678DNJKNKwkd8: {email: "...", name: "..."}
  }
}
```

Přihlášený uživatel může měnit jen svoje data, kde se shoduje jeho UID (*auth.uid*) s uzlem v databázi.

```
{
  "rules": {
    "users": {
      "$uid": {
        ".write": "$uid === auth.uid"
      }
    }
  }
}
```

Pravidla *.write* a *.read* se aplikují i na potomky, dokonce jejich pravidla i přepisují.

Pravidlo *.validate* se kontroluje před vložením záznamu do databáze, pokud záznam nevyhovuje pravidlu, neuloží se.

```
{
  "rules": {
    "users": {
      "$uid": {
        ".write": "$uid === auth.uid",
        "name": {
          ".validate": "newData.isString() && newData.val().length < 25"
        }
      }
    }
  }
}
```

Jméno uživatele může změnit jen jeho vlastník, a pokud typ nových dat bude String a délka řetězce bude menší než 25 znaků. *newData* je další z proměnných.

Posledním pravidlem je *.indexOn*. Indexování slouží k optimalizaci řazení. Firebase automaticky indexuje názvy uzlů, ale může se nastavit, aby indexoval i další atributy [12]. Např. u databáze:

```
{
  users: {
    xnuf7af8nfi2fbFfw211: {email: "...", name: "..."},
    dbjdb6dtw7DWD78wdDV7: {email: "...", name: "..."},
    aodn878678DNJKNKwkd8: {email: "...", name: "..."}
  }
}
```

jsou indexována jednotlivá uid (jakožto názvy uzlů), ale pokud chceme přidat indexy na jméno (name), tak to můžeme udělat přes pravidla:

```
{
  "rules": {
    "users": {
      ".indexOn": ["name"]
    }
  }
}
```

4.2.2 Firebase Storage

Firebase Storage je cloudové úložiště uživatelských dat, jako jsou videa nebo fotografie. Struktura je podobná jako u Firebase Database, jenom jednotlivé uzly v kořenovém stromě neobsahují objekty, ale adresáře a v nich se nachází samotné soubory.

Při vkládání souboru do úložiště, Firebase Storage vrací url, přes které se dají data stáhnout. To si stačí uložit do databáze, a potom přes něj k datům přistupovat.

Data jsou chráněná pomocí pravidel, stejně jako ve Firebase databázi.

4.3 Implementace řešení

Aplikace je implementovaná v jazyce TypeScript [25], což je nadstavba nad jazykem JavaScript, rozšířený např. o statické typování (a typovou kontrolu), třídy, rozhraní, výčtové typy aj.

Aplikace se skládá ze tří obrazovek. První slouží k přihlášení do systému, druhá k výběru testu, na které má uživatel práva, a třetí je pracovní, skládá se z mnoha oken (komponent) a slouží k analýze vybraného testu.

4.3.1 Přihlášení a odhlášení z Firebase

Firebase vyvíjí společnost Google, proto k přihlášení potřebujeme Google účet. Ke komunikaci mezi Firebase a Angularem slouží knihovna *Angularfire2*. Knihovna obsahuje několik modulů, k autorizaci slouží *AngularFireAuthModule*, který obsahuje service s funkcemi pro přihlášení/odhlášení.

Funkce zobrazí popup okno s výběrem Google účtů, kde je možnost si vybrat uživatelský email a po zadání hesla, se uživatel přihlásí do databáze Firebase. Odteď bude mít uživatel přístup jen k datům, které byly vytvořeny s tímto účtem, nebo vlastník dat přidal práva, pro čtení anebo psaní, tomuto emailu.

Po přihlášení se aplikace přesměruje na stránku s výběrem testu. Pokud se uživatel chce odhlásit, nebo změnit účet, tak použije tlačítko *Logout* v hlavičce obrazovky. Po odhlášení se aplikace přesměruje na přihlašovací stránku.

Každý registrovaný uživatel dostane od Googlu svůj identifikátor - UID. V databázi jsou nastavena práva, kde se kontroluje, jestli přihlášený uživatel má přístup k datum, když odešle požadavek o data.

4.3.2 Výběr testu

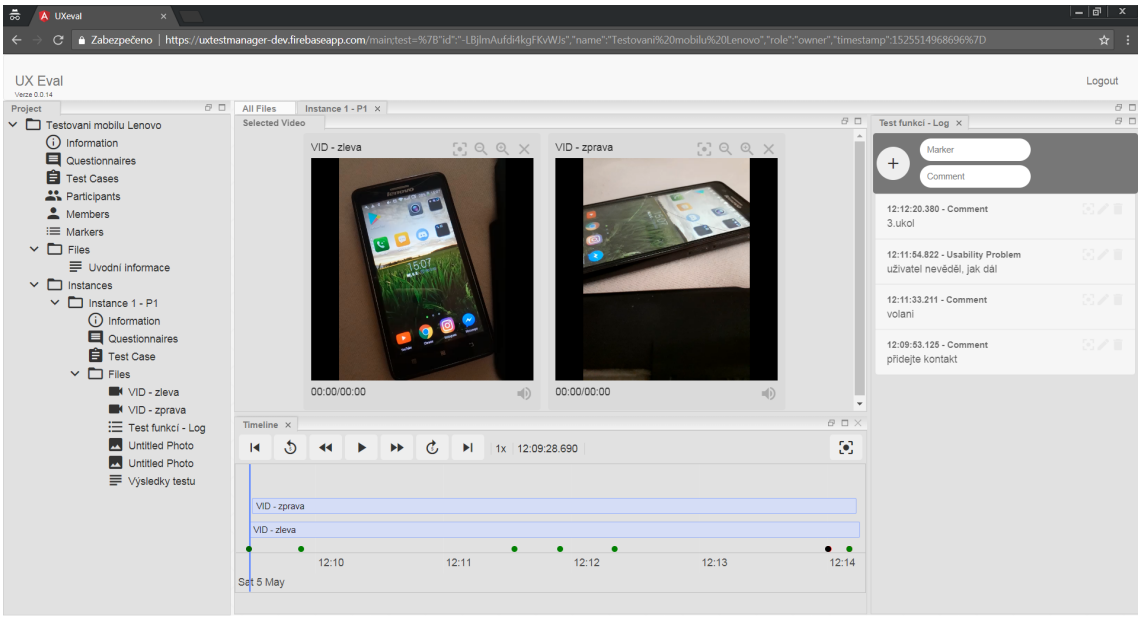
Po přihlášení se uživateli zobrazí seznam testů, které má k dispozici. Pokud byl uživatel přidán do cizího testu, tak k němu má práva v nabídce ho uvidí také. Po výběru testu se aplikace přesměruje na hlavní, pracovní stránku.

4.3.3 Hlavní okno

Jelikož aplikace má sloužit pro analýzu dat, tak je lepší, aby uživatel viděl co nejvíce informací najednou. Inspiroval jsem se některými IDE [6][20], kde je layout obrazovky rozdělen na menší okna (viz Obrázek 4.1), která zobrazují různý typ dat a vykonávají jinou funkci. Jednotlivá okna jsou implementovaná jako samostatné komponenty a data si vyměňují pomoci service.

Díky knihovně *Golden layout*[8] si může uživatel okna roztahovat, přemisťovat nebo zavírat a tím si prostředí upravovat tak, jak to potřebuje.

Uživatelské prostředí se rozděluje na 3 svislá okna. Vlevo je okno, které zobrazuje stromovou strukturu testu. Vpravo je okno pro zobrazení logů a jejich vytváření během analýzy videa. V prostředním okně můžeme otevřít a zobrazit všechna ostatní data. V dalších podkapitolách popíši jednotlivá okna a jejich funkci.



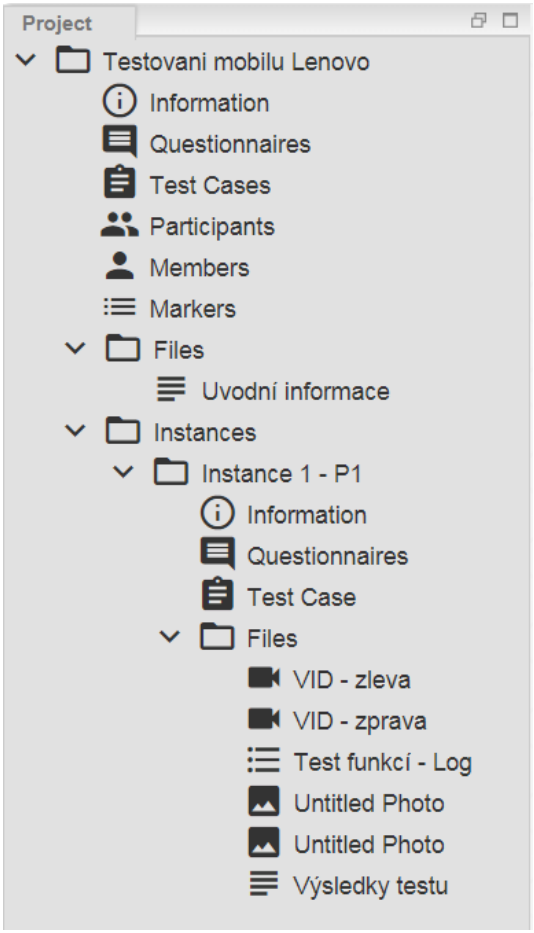
Obrázek 4.1: Hlavní stránka

4.3.3.1 Okno se strukturou testu

Na levé straně hlavního okna se nachází komponenta, která zobrazuje strukturu testu (viz Obrázek 4.2). Test se skládá z několika typů souborů:

- **Informace o testu (Information)** - informace o testu ve formě textu.
- **Dotazníky (Questionnaires)** - seznam pre-test a post-test otázek.
- **Testovací scénáře (Test Cases)** - seznam testovacích scénářů spolu s jejich úkoly.
- **Participanti (Participants)** - seznam účastníků testu, kteří testují produkt.
- **Členové testu (Members)** - lidé, kteří mají přístup k testu. Může se jednat o člena (nemá přístup k souborům a instancím) nebo vlastníka (má stejná práva jako vlastník testu).
- **Markery (Markers)** - seznam značek, které se volí při vytváření logu. Marker slouží pro rychlejší a jednodušší popis nálezu.
- **Soubory (Files)** - jedná se o adresář, který když se rozbalí, tak zobrazí soubory, které jsou k testu přiložené. Test může mít čtyři typy souborů - *log*, *text*, *photo* a *video* (viz kapitola 2.6.1).
- **Instance (Instances)** - opět se jedná o adresář, který obsahuje seznam instancí. Instance se vztahuje vždy k jednomu participantovi a k jednomu testovacímu scénáři, může se v ní také doplnit odpovědi na pre-test a post-test otázky. Instance taky obsahuje adresář *Files*, kde jsou uloženy soubory, které náleží jenom dané instanci.

Při kliknutí na soubor se vytvoří komponenta v příslušném okně (podle typu souboru) a načtou se data, která obsahuje. Každý typ souboru (každý řádek v seznamu) má ještě menu s "dalšími možnostmi", jako např. přejmenování souboru, nebo vytvoření nového souboru ve složce *Files* apod.



Obrázek 4.2: Struktura testu

4.3.3.2 Okno pro zobrazení logů

Okno se nachází na pravé straně a slouží k zobrazení logů, které vytvářejí během analýzy videa. Log soubor se skládá z časové značky, markeru a komentáře (viz Obrázek 4.3).

Log slouží jako rychlá a krátká poznámka o nálezu. Pomocí časové značky se může rychle najít na časové ose a uživatel se může na daný moment znova podívat. Při vytvoření logu se časová značka vezme aktuálního času na časové ose, aby byl log synchronizovaný s videem a celým testem. Marker se může vybrat z předdefinovaných a uložených v testu, nebo se může napsat jakýkoliv text. Komentář může být prázdný nebo jakýkoliv text.

Logy se zobrazují pod sebou, navrchu jsou nejnovější logy, čili jsou seřazeny podle času. Log je možné upravovat a taky pomocí tlačítka se může vyhledat na časové ose, aktuální čas

na ose se nastaví na čas logu, tím pádem se i videa přetočí na čas logu.

Obrázek 4.3: Logy instance

4.3.3.3 Okno s časovou osou

Prostřední okno je univerzální, buď zobrazuje časovou osu s vybranými video záznamy nebo v něm můžeme zobrazit jakákoliv jiná data, kromě logů.

Test použitelnosti se většinou nahrává na videozáznam, může jich být i více, např. z jiného úhlu. Proto je v aplikaci možnost zobrazit více videí a analyzovat je zároveň (viz Obrázek 4.4). Videa se vyberou ze seznamu souborů instance.

Ve spodním okně se nachází časová osa (Timeline). Ta je vizualizovaná knihovnou *Vis.js* a zobrazuje všechny soubory instance, ty jsou barevně odlišené. Taky je vertikální modrou čarou zobrazený aktuální čas osy. Dále jsou nad osou tlačítka na ovládání videí a osy. Je to podobné jako u přehrávačů médii, kde jsou tlačítka na přehrání, pausy, zrychlení, zpomalení, skok o 5 sekund dopředu nebo dozadu, skok na začátek nebo na konec testu.

Ovládací tlačítka ovládají aktuální čas osy a podle něj se synchronizují videa. Čas se aktualizuje každých 100 ms (1 tik) a o tuto hodnotu se posune svislá osa. Přitom se pomocí metody *Subject<T>.next()* pošlou data o aktuálním čase, informací o tom jestli se osa právě přehrává, jakou rychlostí a id instance.

```
if (this.timeline) {
    this.currentTime += this.frame * this.speedRate;
    this.timelineTimeSubject.next({
        time: this.currentTime,
        play: this.isPlaying,
        speed: this.speedRate,
        instId: this.instance.info.id
    });
    this.timeline.setCustomTime(this.currentTime, "mainBar");
}
```

Data odeslaná pomocí metody *next()* jsou přijaty v komponentě, která zobrazuje videa a očekává je, protože si o ně požádala během vytvoření komponenty.

```
this.timelineTimeSubject.asObservable().subscribe((data: any) => {
    // funkce, kde se ovládají a synchronizují videa
});
```

Časovou osu se nemůže aktualizovat moc často, protože by to procesor nestíhal, a proto časová osa neaktualizuje čas videa, ale kontroluje se příznak *isPlaying* a pokud je video zastavené, tak se začne přehrávat a obráceně. Pokud je na videu nastavený jiný čas, tak se nastaví na ten správný, podle časové osy. Každých 100 ms tedy dochází ke kontrole jestli časy videí sedí s časovou osou. Hodiny video přehrávače jsou ale přesnější, zatímco hodiny webového prohlížeče ne. 100 ms se dá odpočítávat např. opakovaným volání JS funkce *setTimeout()*.

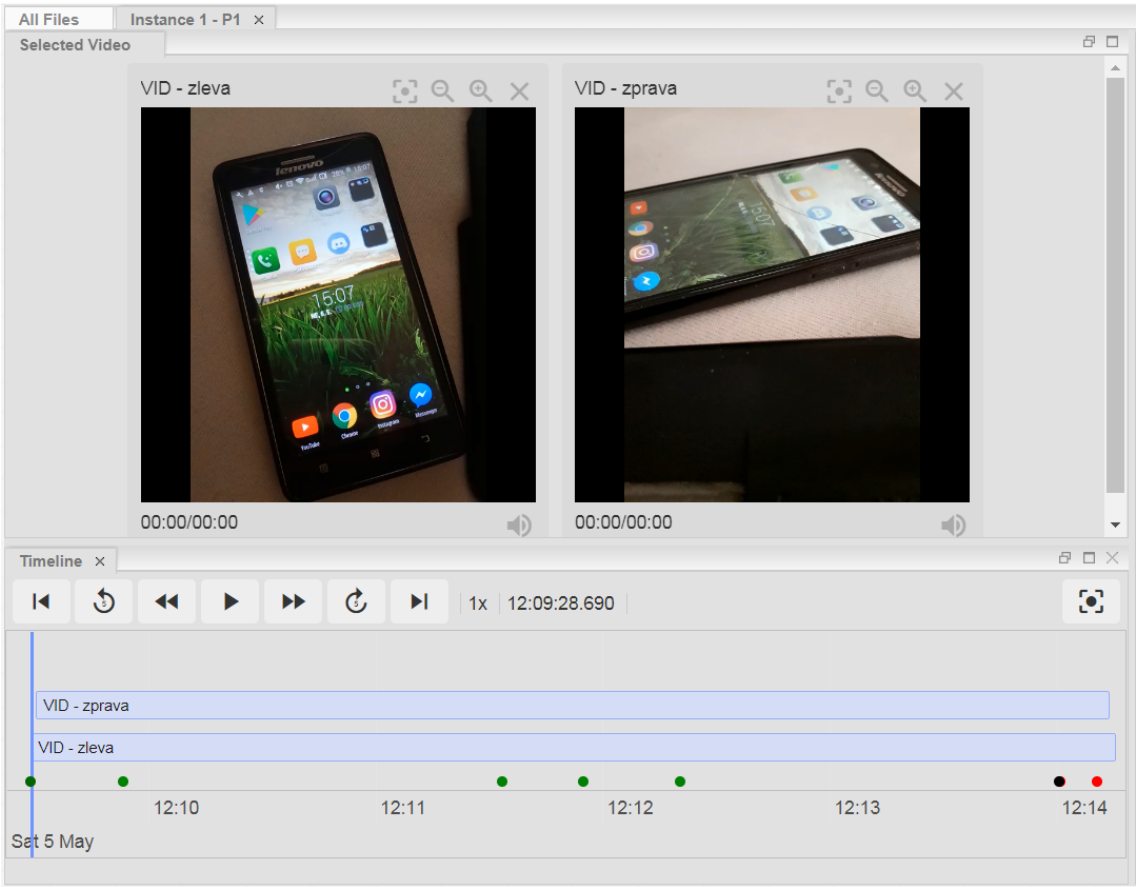
```
private go() {
    // telo funkce
    setTimeout(() => {this.go();}, 100);
}
```

Jelikož ale hodiny browseru nejsou úplně přesné, tak se funkce *go()* zavolá např. za 110 ms. Chyba se bude akumulovat a po nějaké době se tak stane, že čas na ose a ve videích se nebude shodovat. Proto jsem použil princip zpětné vazby, když funkce *next()* pošle aktuální čas osy a videa si ho kontrolují, tak si spočítají o jakou hodnotu se liší se svým časem, tuto hodnotu uloží do sdílené proměnné (pomocí service a public proměnné) a v dalším tiku se časovač nastaví o tuto hodnotu méně.

```
setTimeout(() => {this.go();}, 100 - diff);
```

Tím se v každým tiku dorovná ztráta z minulého tiku. Hodiny stále nebudou přesně synchronizované, ale k akumulaci chyby nedochází tak rychle. Potom jsem ještě přidal kontrolu, že pokud je rozdíl časů větší než pevně daná hodnota (např. 1 s), tak se čas videa nastaví na aktuální čas časové osy.

Tímto způsobem zůstávají všechna videa synchronizovaná s časovou osou a se sebou navzájem.



Obrázek 4.4: Časová osa a 2 analyzovaná videa

Kapitola 5

Testování

V této kapitole budu probírat testování aplikace. Testoval jsem software, hlavně ukládání dat do databáze a taky jsem provedl uživatelské testování s potenciálními uživateli. Důležité pro mě bylo hlavně uživatelské testování použitelnosti aplikace, protože o tom je i táto práce. Nalezlo se během ní několik chyb, které mě během vývoje nenapadly a stejně tal jsem obdržel i nějaké nápady na nové funkce.

5.1 Uživatelské testování

Co je test použitelnosti jsem popsal v kapitole 2.1. Ten jsem taky použil k otestování mé aplikace. Díky testům s uživateli bylo nalezeno několik chyb, které mě během vývoje nenapadly a také participanti přispěli s nápady a vylepšeními, které by se dali v budoucí verzi implementovat.

Cílem testování bylo zjistit, jestli je ovládání aplikace intuitivní a jestli poskytnuté nástroje jsou dostatečné pro správu a analýzu dat z testů použitelnosti.

5.1.1 Výběr participantů

Aplikace je pomůcka pro výzkumníky, kteří provádí testy použitelnosti, a poté je analyzují. Je to velmi specifická aplikace, a proto i výběr participantů je úzký. Vybíral jsem participanty z řad studentů ČVUT oboru Interakce člověka s počítačem (HCI[1] - jedná se o obor, který zkoumá interakce a komunikaci mezi člověkem a počítačem) a studenty předmětu TUR (Testování uživatelského rozhraní), kde v obou případech studují a provádí testy použitelnosti. Participanti tedy mají zkušenosti s prováděním a analýzou testů. Jelikož další kritéria nejsou požadována, tak jsem ani nedělal screener.

Testoval jsem 5 participantů, 4 z oboru HCI a jednoho z předmětu TUR. Všichni někdy prováděli test použitelnosti ať už v rámci semestrální nebo diplomové práce.

[1] Human Computer Interaction

5.1.2 Seznam úkolů

K otestování důležitých funkcí a problémových míst jsem vytvořil testovací scénář s deseti úkoly. Uživatel prověří vytváření a úpravu dotazníku, test case, logů, souborů a práci s videi. Participant dostal úkoly a připravené materiály, aby nemusel data vymýšlet.

1. Včera proběhl test mobilního telefonu Lenovo A536, přihlaste se a otevřete test, abyste ho mohl analyzovat (email: uxeval.test1@gmail.com, heslo: Heslo123).
2. Výzkumník ze včerejšího testování zapomněl do aplikace vložit pre-test a post-test dotazníky a testovací scénáře. Podle připravených materiálů je do testu vložte.
3. Jeden z pomocných výzkumníků se k testu nemůže dostat, nasdílejte mu test, aby mohl data také vidět. Email výzkumníka je dimatest01@gmail.com, dejte mu práva "owner".
4. Přejdeme k instanci participanta P1 a k analýze jeho dat. Vyplňte odpovědi na pre-test dotazník participanta P1:
   - Otázka 1 - "ano"
   - Otázka 2 - "dotykový"
5. Participant testoval testovací scénář "Test funkcí - Android", výzkumník ale do instance participanta zadal špatný test case, opravte to.
6. Chystáme se analyzovat videa z testování, v Instanci participanta vytvořte Log soubor, kam později budete psát poznámky a komentáře, Log pojmenujte "Test funkcí - Log".
7. Analyzujte instanci participanta - vyberte 2 videa, použijte okno "Timeline"k ovládání videí a analýze, poznámky vkládejte do vytvořeného logu (používejte různé Markery).
8. Výzkumník zapomněl vložit fotografii mobilního telefonu, který se testoval. Vložte fotografii mobilu z počítače - lenovo.jpg.
9. V instanci vytvořte textový soubor, kam zapíšete shrnutí o výsledku testování, soubor pojmenujte "Výsledky testu". Starý textový dokument "Poznámka k testu"můžete smazat.
10. Rád byste se ještě vrátil k jednomu z videu a zkontroloval ho, chcete ho otevřít samostatně, bez závislosti na časové ose.

5.1.3 Testovací prostředí

Testovalo se na notebooku Windows 7 a prohlížeči Google Chrome verze 66, verze aplikace byla 0.0.15.

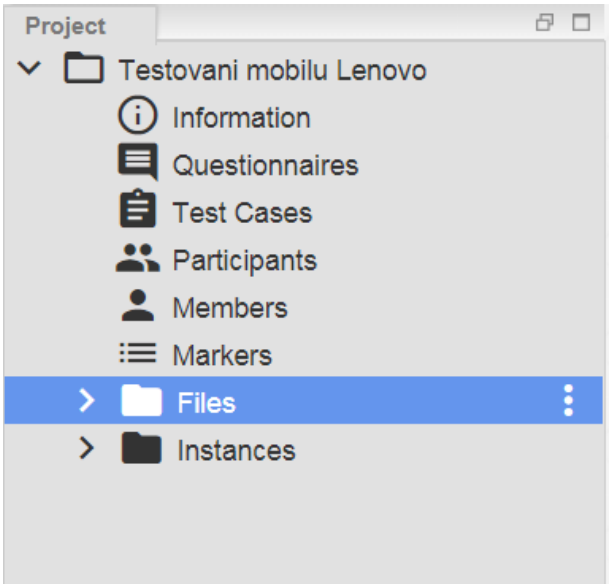
5.1.4 Sběr a vyhodnocení dat

Nasbíral jsem několik nálezu a také participanti nabízeli další nápady, jak by se dala aplikace vylepšit. Níže je seznam nálezů a jejich možné řešení. Zmiňuji nálezy, na kterých se shodlo aspoň 3 z 5 participantů a jsou seřazeny od nejvíce důležitých a kritických po nejméně.

5.1.4.1 Další možnosti u souborů

Nález

Ve stromu projektu (testu) každý soubor (řádek), pokud se na něj najede myší, tak zobrazí ikonu menu (viz obrázek 5.1). Po kliknutí na ni, se zobrazí další možnosti (např. přejmenování, smazání, otevření apod.). Jelikož se ikona zviditelní až po přejetí myši, tak participanti (5/5) nevěděli o této ikoně a o dalších funkcí a ani si jí nevšimli i po chvilce hledání.



Obrázek 5.1: Zobrazení dalších možností až po najetí myši

Řešení

Jednoduchým řešením je zviditelnit ikony u všech souborů. Graficky to sice nebude vypadat tak dobře, ale funkčnost je přednější.

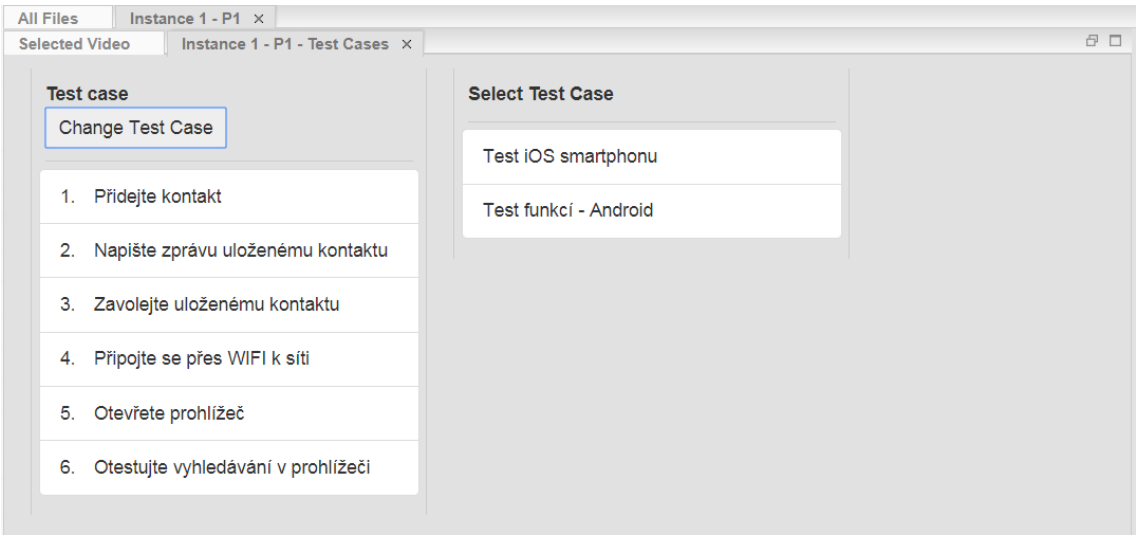
5.1.4.2 Výběr testovacího scénáře instance

Nález

U testovacího scénáře (Test Case) v instanci participanta, je vybraný testovací scénář dané instance a zobrazuje se jenom seznam úkolů (Tasks). Participanty (4/5) mátlo, že rozložení obrazovky je jiné než při vytváření Test Case v sekci testu a není napsaný název testovacího scénáře, takže nevěděli, který je zvolený.

Řešení

Bylo by vhodné obrazovku s výběrem Test Case v instanci udělat stejnou, jako v sekci testu. Napsat název vybraného Test Case a např. zobrazit ostatní, ale s tmavším pozadí, aby bylo vidět, že jsou i jiné na výběr.



Obrázek 5.2: Matoucí výběr testovacího scénáře instance

5.1.4.3 Viditelnost ikon

Nález

Většina tlačítek jsou tvořena jenom ikonami bez textu, ty ale jsou moc průhledné a participanti si jich nevšimli na první pohled (viz obrázek 5.3).

Obrázek 5.3: Vkládání logů s ikonami pro editace a smazání vpravo

Řešení

Ikony je potřeba ztmavit a zvětšit, aby na šedivém podkladu byly lépe vidět.

5.1.4.4 Zarolování uzlů stromu testu

Nález

Pří změně souboru dojde k aktualizaci stromu testu a dojde k zarolování uzlů, které obsahují potomky (např. uzel *Files* a *Instances*). Uživatel poté musí uzle opět odrolovat.

Řešení

Jedná se o programátorskou chybu a je velmi frustrující a otravná. Uzly by si měly pamatovat jestli jsou odrolované nebo ne.

5.1.4.5 Výchozí název vytvořeného souboru

Nález

Během vytvoření souboru se mu nastaví nějaký výchozí název, a potom ho může uživatel změnit. To přišlo participantům otravné a uvítali by možnost, kdy by mohli název napsat hned při vytvoření souboru. Zajímavou poznámkou taky bylo, že při přejmenování ho musí nejdříve najít, což může být zdlouhavé, pokud instance obsahuje hodně souborů.

Řešení

Při žádosti o vytvoření souboru, by měla aplikace zobrazit okno, kde by uživatel napsal název souboru, a poté by se vytvořil.

5.1.4.6 Vytvoření souboru

Nález

Vytvoření nového souboru (log, textový dokument, fotografie, video) se provádí ve složce *Files* ve stromu projektu (testu), kde jsou také umístěné. Tam ale participanti (3/5) nehledali a hledali přidání souboru v uzlu *Instances* (podle úkolu měli do instance přidat soubor).

Řešení

Jelikož soubory patří jen dané instanci, tak se můžou v okně, se stromem testu, přidat přes uzel *Instances* tak i přes uzel *Files* (pomocí ikony pro "další možnosti").

5.1.4.7 Chybí placeholder v textovém poli

Nález

Během vyplňování údajů (dotazník, testovací scénář a úkoly apod.) si participanti nebyli jisti, co mají do textového pole psát.

Řešení

Každé textové pole by mělo obsahovat placeholder, s popisem co do pole patří.

5.1.4.8 Automatické ukládání

Nález

Participanti si nebyli jisti, jestli při změně textu (např. text otázky v dotazníku) se změny uložili na server, nebo musí ještě něco potvrdit.

Řešení

Při změně jakýkoliv dat by mela aplikace upozornit uživatele, že byly změny úspěšně uloženy.

5.1.4.9 Nápady participantů

Během testování participanti nabízeli nápady na nové funkce a vylepšení. V budoucí verzi bych je rád implementoval, protože mi přijdou zajímavé a relevantní.

- Přidat klávesové zkratky.
- Zobrazit alert pokud odcházím ze stránky a neuložím změny.
- Možnost přidávání souboru v okně Timeline.
- Otevírat soubor přesouváním - drag & drop.
- Měnit pořadí otázek a úkolů pomocí drag & drop.
- Při napsání nového Markeru v logu ho rovnou přidat mezi Markers v testu.
- Markery barevně odlišovat a mít možnost barvu nastavovat. Nyní se barevně odlišují jen typy souborů.
- Zvýraznit Marker (v otevřeném log souboru), který je vybraný v Timeline.

Dále mě během vývoje napadly další funkce, které by se mohly do aplikace přidat.

- **Elektronická tužka** - kterou by se dalo malovat do fotografie, např. pokud by výzkumník chtěl upozornit na nějakou její část.
- **Screenshot z videa** - k videu bych rád přidal tlačítko, které by vytvořilo screenshot videa v aktuálním čase a fotografii uložila do souborů.
- **GPS** - jelikož se testy použitelnosti často dělají i na ulici (např. navigace s nevidomými), tak bych rád do aplikace přidal mapu (např. pomocí Google Maps [17]), která by zobrazovala aktuální polohu participanta (musí být k dispozici souřadnice v databázi).

5.2 Softwarové testování

S rostoucí aplikací je těžší ji testovat ručně, a proto Angular nabízí end-to-end testování [3]. Unit testy můžeme psát přímo v našem projektu a nejsou potřeba další nástroje. Všechny potřebné věci pro testování se vytvoří už při vygenerování projektu.

K testování Angular používá Node.js knihovnu Protractor [21], který používá Selenium WebDriver [23] k ovládání prohlížeče a simulování interakce člověka s aplikací.

5.2.1 Struktura testu

V souboru */e2e/app.e2e-spec.ts* se nachází funkce k testování. Scénář testu se nachází ve funkci *describe()*, v něm probíhají všechny testy. Testy jsou volány pomocí funkce *it()*, ve kterých se provádějí příkazy, např. aby se zmáčklo tlačítko podle ID, nebo se vložil text to textového pole, a nakonec se pomocí funkce *expect()* prověří, zda se data shodují s těma, které očekáváme. Před všemi testy se volá funkce *beforeAll()* a před každým testem zvlášť se volá funkce *beforeEach()* (obdobně se nakonec vola funkce *afterEach()* a *afterAll()*), kde si můžeme např. nastavovat proměnné apod. Ve výsledku může kód vypadat například takto:

```
describe('UXeval testing', () => {

  // trida s pomocnymi funkcemi
  let page: AppPage;

  beforeAll(() => {
    page = new AppPage();
    browser.driver.manage().window().maximize();
  });
  beforeEach(() => {
    browser.sleep(1000);
  });

  it('should fill email', () => {
    page.navigateTo("/home");
    let el: ElementFinder = element(by.id("emailInput"));
    el.sendKeys("uxeval.test1@gmail.com");

    expect(el.getText()).toEqual("uxeval.test1@gmail.com");

  });

  it('should choose test', () => {...});
});
```

5.2.2 Výsledky

Softwarovým testováním jsem nenalezl takové nedostatky a chyby jako s uživatelským testováním, spíše jsem kontroloval funkce aplikace a jestli správně funguje komunikace s databází. Některé části programů se opakují, např. vytváření participantů a vytváření markerů, takže i test je stejný. Díky tomu se dala nově naprogramovaná funkčnost lehce otestovat.

Velké chyby jsem nenašel, spíše se jednalo o menší programátorské chyby, které jsem hned opravil.

Kapitola 6

Závěr

V této diplomové práci jsem se zabýval problematikou testů použitelnosti, které jsou důležitou součástí vývoje nového produktu. Popsal jsem jak se připravují, jak probíhají, kde se dá testovat a jak můžeme data z testů sbírat. Analyzoval jsem několik aplikací pro sběr a analýzu dat (UX Tester, IVE, Morae, Dartfish) a některými prvky a funkcemi, které mi přišli zajímavé a užitečné, jsem se inspiroval. Vytvořil a popsal jsem případy užití, které by měla aplikace splňovat. Podle nich jsem navrhl dva prototypy pomocí softwaru Balsamiq a jeden z prototypu jsem implementoval.

Aplikaci UX Eval jsem napsal pomocí frameworku Angular 5 v programovacím jazyce TypeScript. Aplikace umožňuje práci s projekty vzniklými v aplikaci UX Tester, což je mobilní aplikace zaměřená na sběr dat z testů použitelnosti v reálném prostředí (např. i na ulici). Aplikace UX Eval se umí napojit na stejnou databázi, Firebase Realtime Database, a pokud se provedou změny v datech v aplikace UX Tester, tak se ihned zobrazí i v této aplikaci.

UX Eval pomáhá výzkumníkům s analýzou dat z testů použitelnosti, umí vytvářet, upravovat a celkově spravovat participanty, dotazníky, testovací scénáře, soubory a instance těchto testů. Důležitá část je práce s časovou osou, na které jsou vidět všechna data, která se dají procházet. Časová osa nabízí různé funkce pro její ovládání, jako zrychlení, zpomalení, skok o 5 vteřin dopředu/dozadu. Díky tomu si výzkumník může test důkladně procházet. Uživatel si může otevřít a prohlížet více videí najednou, analyzovat je a vytvářet log s poznámkami v požadovaný čas.

Nakonec jsem provedl testy použitelnosti s uživateli. K testu se přihlásilo pět studentů oboru Interakce člověka s počítačem, takže jsem mohl testovat s lidmi, kteří mají zkušenosti s testy použitelnosti. Díky nim jsem objevil několik logických chyb, ale hlavně jsem od nich dostal zpětnou vazbu s připomínkami a nápady.

V rámci budoucí práce bych rád nápady z testování realizoval a taky naprogramoval některé další funkce, které mě napadly během vývoje (blíže jsem je popsal v podkapitole 5.1.4.9). Sice nejsou nezbytně nutné, ale můžou pomoct v analýze testů použitelnosti.

Literatura

[1] ANGULAR. *Angular* [online]. [cit. 20. 04. 2018]. Dostupné z: <https://angular.io/>.

[2] ANGULAR. *Angular Binding* [online]. [cit. 20. 04. 2018]. Dostupné z: <https://angular.io/guide/template-syntax#binding-syntax-an-overview>.

[3] ANGULARJS. *E2E Testing* [online]. [cit. 11. 05. 2018]. Dostupné z: <https://docs.angularjs.org/guide/e2e-testing>.

[4] BARNUM, C. *Usability testing essentials : ready, set...test!* Burlington, USA : Elsevier Inc., 1 edition, 2011. ISBN 978-0-12-375092-1.

[5] B.V., A. *Vis.js* [online]. [cit. 25. 04. 2018]. Dostupné z: <http://visjs.org/>.

[6] DARTFISH. *Dartfish* [online]. [cit. 20. 03. 2018]. Dostupné z: <http://www.dartfish.com/>.

[7] DAVID EAST, J. C. *Angularfire2* [online]. [cit. 11. 05. 2018]. Dostupné z: <https://github.com/angular/angularfire2>.

[8] GMBH. *GoldenLayout* [online]. [cit. 11. 05. 2018]. Dostupné z: <http://golden-layout.com/docs/>.

[9] FIREBASE, I. *Firebase* [online]. [cit. 13. 04. 2018]. Dostupné z: <https://firebase.google.com/>.

[10] FIREBASE, I. *Firebase Realtime Database* [online]. [cit. 25. 04. 2018]. Dostupné z: <https://firebase.google.com/docs/database/>.

[11] FIREBASE, I. *Firebase Hosting* [online]. [cit. 20. 04. 2018]. Dostupné z: <https://firebase.google.com/docs/hosting/>.

[12] FIREBASE, I. *Index Your Data* [online]. [cit. 25. 04. 2018]. Dostupné z: <https://firebase.google.com/docs/database/security/indexing-data>.

[13] FIREBASE, I. *Firebase Products* [online]. [cit. 20. 04. 2018]. Dostupné z: <https://firebase.google.com/products/>.

[14] FIREBASE, I. *Understand Firebase Realtime Database Rules* [online]. [cit. 25. 04. 2018]. Dostupné z: <https://firebase.google.com/docs/database/security/>.

[15] FIREBASE, I. *Structure Your Database* [online]. [cit. 25. 04. 2018]. Dostupné z: <`https://firebase.google.com/docs/database/web/structure-data`>.

[16] MALÝ, I. *Analysis of Usability Tests with Context Model.* Praha : České vysoké učení technické v Praze, 2008.

[17] MAPS, A. *Angular Google Maps* [online]. [cit. 20. 05. 2018]. Dostupné z: <`https://angular-maps.com/`>.

[18] NIELSEN, J. *Why You Only Need to Test with 5 Users* [online]. 2000. [cit. 19. 03. 2018]. Dostupné z: <`https://www.nngroup.com/articles/why-you-only-need-to-test-with-5-users/`>.

[19] NPM, I. *npm* [online]. [cit. 25. 04. 2018]. Dostupné z: <`https://www.npmjs.com/`>.

[20] POŽIVIL, J. *Integrated Visualization Environment for Interactive Analysis of User Activity Logs.* Praha : České vysoké učení technické v Praze, 2009.

[21] PROTRACOTR. *Protracotr* [online]. [cit. 11. 05. 2018]. Dostupné z: <`https://github.com/angular/protractor`>.

[22] RUBIN, J. – CHISNELL, D. *Handbook of Usability Testing.* Indianapolis : Wiley Publishing, Inc., 2 edition, 2008. ISBN 978-0-470-18548-3.

[23] SELENIUM. *Selenium* [online]. [cit. 11. 05. 2018]. Dostupné z: <`https://github.com/seleniumhq/selenium`>.

[24] TECHSMITH. *Morae* [online]. [cit. 20. 03. 2018]. Dostupné z: <`https://www.techsmith.com/morae.html`>.

[25] TYPESCRIPT. *TypeScript* [online]. [cit. 13. 04. 2018]. Dostupné z: <`http://www.typescriptlang.org/`>.

[26] ŠPATNÝ, J. *Mobile applications for management and data collections in field usability studies.* Praha : České vysoké učení technické v Praze, 2017.

Příloha A

Přiložené přílohy

Jako přílohy této práci je přiložena elektronická verze tohoto dokumentu a adresář se zdrojovými kódy webové aplikace.